Universitetet i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation: Master of Science in Computer Science / Reliable and Secure Systems	Spring semester, 2021 Open/~~Confidential~~
Author(s): Eirik Solland Haraldsen, Karl Meisland Østrådt	
Program coordinator: Tomasz Wiktorski Supervisor(s): Hallgrim Ludvigsen (Logtek AS)	
Title of master's thesis: Hybrid Human/Machine Interpretation of Well Logs using Deep Learning	
Credits: 30	
Keywords: Machine Learning, Deep Learning, Anomaly Detection, Time Series, Petrophysics	Number of pages: 119 + supplemental material/other: 17 Code available on GitHub Stavanger June 15. 2021

Abstract

Exploration and production wells in the oil and gas industry produce a vast amount of logging data. All petroleum data from the Norwegian continental shelf must be quality controlled and reported in a structured manner to the Norwegian Petroleum Directorate. Structured data of high quality opens up the possibility of exploring the use of machine learning in the field. Machine learning applications have already improved the efficiency of existing systems, and in some cases replaced them entirely.

This project aims to assist human interpretation of well logs with the use of deep learning methodologies. Quality control and identification of zone boundaries are two time-consuming workflows that could benefit from deep learning. We propose a deep learning based approach for predicting candidate points of interest across a well log. The proposed approach aims to improve the efficiency of the petrophysical workflow by limiting data interpretation to fewer sections. We develop a preprocessing pipeline for well log data, and implement four deep learning algorithms. Additionally, we develop two approaches for model evaluation.

The first evaluation approach achieve an F_1 score and Matthews correlation coefficient (MCC) of 0.35 and 0.32 respectively. The second approach achieve a better performance with an F_1 score and MCC of 0.54 and 0.49 respectively. The results suggests that our second approach is capable of identifying erroneous data and lithology boundaries. This indicates that our model can be a beneficial addition to the interpretation of well logs.

Acknowledgements

We would like to thank our supervisor Tomasz Wiktorski for his advice, feedback and guidance throughout our work.

We would also like to thank Hallgrim Ludvigsen for his continuous support, guidance and motivation of this thesis.

A strong gratitude goes out to everyone at Petroware and Logtek for their support. This thesis would not have been possible without their assistance with understanding the petrophysical background.

We are also grateful of Logtek AS for the opportunity to work with them, and for providing us with an abundance of well log data.

We thank Equinor AS, the former Volve license partners ExxonMobil Exploration and Production Norway AS and Bayerngas (now Spirit Energy) for permission to use the Volve dataset, and to the many persons who have contributed to the work here. Please visit data.equinor.com for more information about the Volve dataset and license terms of use.

Finally, Eirik would like to thank his family and significant other for their continuous support and encouragements throughout his Master's degree. Karl would like to thank his family and friends for their encouragements throughout this thesis.

Contents

Abbreviations

AC	Acoustic Compression
ACC	Accuracy
ACS	Acoustic Shear
AE	Autoencoder
BCE	Binary Cross-Entropy
BHA	Bottom Hole Assemply
BPTT	Back-Propagation Through Time
BS	Bit Size
CALI	Caliper
CNN	Convolutional Neural Network
DeepAnT	Deep learning approach for unsupervised Anomaly detection in Time series data
DEN	Density
DENC	Density Correction
DEPTH	Measured Depth
DLIS	Digital Log Interchange Standard
ED	Euclidean distance
EWL	Electric Wireline Logging

F	F-score
FFNN	Feed-Forward Neural Network
FN	false negative
FP	false positive
GR	Gamma Ray
LAS	Log ASCII Standard
LIS	Log Interchange Standard
LOCF	Last Observation Carried Forward
LSTM	Long Short-Term Memory
LWD	Logging While Drilling
MAE	Mean Absolute Error
MAR	Missing at Random
MCAR	Missing Completely at Random
MCC	Matthews Correlation Coefficient
ML	Machine Learning
MSE	Mean Square Error
N	actual negative
NCS	Norwegian continental shelf
NDR	National Data Repository
NEU	Neutron Porosity
NMAR	Not Missing at Random
NOCB	Next Observation Carried Backward
NPD	Norwegian Petroleum Directorate
P	actual positive

PEF Photoelectric factor

PN predicted negative

POI Point(s) of Interest

PP predicted positive

PPV Positive Predicted Value

RDEP Deep Resistivity

ReLU Rectified Linear Unit

RMED Medium Resistivity

RNN Recurrent Neural Network

TN true negative

TP true positive

TPR True Positive Rate

VAE Variational Autoencoder

Chapter 1

Introduction

1.1 Motivation

The recent explosion in technological advancements have resulted in huge datasets of information from oil and gas exploration and production wells. The existence of large, structured datasets of high quality has started a trend of Big Data analysis in the petroleum industry [1]. This type of data also opens up the possibility of exploring the use of machine learning in the field. Exploring machine learning might prove to be massively beneficial for the oil industry, considering it has already revolutionized many other fields [2]. Researchers at IBM have already started exploring how artificial intelligence can help reduce downtime related costs by monitoring and predicting equipment failure [3]. However, a huge problem for researchers is getting access to the large amounts of data. The majority of well log data is confidential and considered market sensitive information. Our collaboration with Logtek AS put us in a unique position where we had access to a large quantity of historical and recent well log data. The motivation behind this project is to explore how we can utilize the data and deep learning to improve the efficiency of systems or workflows within the company.

1.2 Problem Definition

With current solutions, the huge amount of well log data is a massive issue. Petroleum engineers and petrophysicists spend over half their time searching for and preparing data before a proper analysis can be performed [4]. The aim of this project is to use deep learning methodologies to assist in human interpretation of well logs. We envision a solution that can contribute to the overall workflow efficiency and provide additional insights to petrophysicists. Through observation and conversation with petrophysicists at Logtek AS, we identified two possible workflows that could benefit from deep learning: boundary zonation and quality control.

The first workflow involves identifying the boundaries of a lithology or lithostratigraphic formation in well logs. Currently, petrophysicists typically look at a handful of logs to identify the correct zone boundaries. This method is quite time-consuming as the interpreter has to manually scour through the different logs and carefully select the boundaries. Our goal is to develop a solution that identifies candidate points for zone boundaries. We envision that the interpreter will still make the ultimate decision, but the aim is to make the workflow more efficient by providing candidate points.

The second workflow is performing quality control to ensure high wellbore data quality. The current procedure is very similar to the one in workflow one, where petrophysicists manually analyze well logs and curve plots. This process is repeated multiple times to ensure all corrections are correct and no errors remain. Our aim is to assist the interpreter responsible for identifying quality issues with the dataset. We would like to provide a solution that identifies areas of the dataset that should be investigated first. We envision this solution can narrow down the number of areas the interpreter need to analyze in detail, effectively reducing the time required to quality control the well log. The process might still need to be repeated multiple times to ensure all errors are corrected.

Both workflows can be viewed as an anomaly detection problem. In workflow one, the definition of an anomaly could be a datapoint that do not follow the current trend in the time series. When the wellbore moves from one lithology to the next, the general trend of the dataset is assumed to change. In workflow two, the definition of an anomaly could be the same

as workflow one, but it could also simply be an abnormal datapoint. We aim to utilize several deep learning algorithms to detect anomalies. Our goal is to develop and compare the performance of both reconstruction and prediction based algorithms.

1.3 Contributions

In this thesis we make the following contributions:

1. Developed a preprocessing pipeline that allows the use of well log data as time series data.
2. We implement and adapt four relevant deep learning algorithms for time series anomaly detection.
3. Adapted and expanded upon anomaly detection method proposed in related work.
4. Worked together with petrophysicists in creating a ground truth containing points of interest with their corresponding depth for the publicly available test dataset.
5. Analyzing and comparing results for two evaluation approaches.

1.4 Thesis structure

Chapter one, **Introduction** provides an overview of the motivation, problem definition and our goals for this project. Additionally, we present the contributions of this work and an outline of the thesis structure.

In chapter two, **Background & Related Works** we introduce some of the necessary petrophysical background needed for this project. The final part of the chapter is an overview of related work of deep learning in the petroleum industry, time series prediction, and anomaly detection.

In chapter three, **Methodology** we present the theoretical background for the methodology used in this project. First, the chapter introduce the file formats of composite well logs and the associated features. Second, important preprocessing steps and some machine learning basics are presented. Third, we present the theory behind the four deep learning algorithms used in this project. The final part of this chapter introduce a few model evaluation metrics.

Chapter four, **Implementation & Materials** presents our implementation of the methodology. The first section of the chapter introduce the training and test datasets, the data collection process, and our preprocessing approach. The second section provides an overview of the implementation of the models. The final section of the chapter explains how we use the output of the models to identify anomalies, and how we evaluate the performance.

In chapter five, **Results, Discussion & Analysis** we present the results for the four deep learning models. We present and discuss the performance of multiple configurations for each model. The final section of the chapter compare the different models.

In chapter six, **Future Work & Conclusion** we present some of the challenges we faced and propose possible future work. Finally, we summarize and conclude the work done in this project.

Chapter 2

Background & Related Works

In this chapter we discuss relevant background information for our project. We introduce important details about the Diskos NDR, Blue Book rulebook, the Norwegian Petroleum Directorate, and important petrophysical and well logging terminology. We also present and discuss previous works related to well log data, time series, anomaly detection and deep learning in this field.

2.1 Petrophysical Background

2.1.1 Basic Petrophysical Properties

The term petrophysics is defined as the study of physical and chemical rock properties and their interaction with fluids [5]. This study is mainly applied in the oil and gas industry for studying the rock and fluid properties of reservoirs. In petrophysics, the measurements are typically obtained from a string of measurement tools that measures various signals along the borehole path. These measurements are used to infer properties that include lithology, porosity, water saturation, permeability, and density. Lithology refers to the physical characteristics of a rock such as color, grain size, composition, and texture. Porosity measures the void spaces in a rock

formation. A porous rock may contain liquids and gas such as water or hydrocarbons in these void spaces. Water saturation describes how much of the pores in a formation is water. Permeability defines the ability of fluids to flow through a rock as a function of time and pressure. A rock with interconnected pores has a higher permeability than a rock with low porosity. Density is defined as mass per unit volume.

2.1.2 Wellbore Logging

The generation of well log data is performed by recording multiple different logs and splicing them together into composite logs. The main types of logs are electrical logs, porosity logs, lithology logs and miscellaneous logs. Electrical logs focuses on measuring resistivity and conductivity. Porosity logs measures density, neuron porosity and sonic (acoustic) waves. Lithology logs measures naturally occurring radiation and spontaneous potential. Miscellaneous logs are logs that don't fit into the three other categories and typically include measurement of caliper, magnetic resonance, and noise logging.

These measurements are usually recorded using the Electric Wireline Logging (EWL) or the Logging While Drilling (LWD) method. EWL utilizes a string of measurement tools that is lowered into the borehole to record petrophysical properties. One disadvantage with this approach is that logging does not occur while drilling. LWD does not have this issue because the well logging tools are integrated with the Bottom Hole Assemply (BHA). This is what allows for logging while drilling. Since the logging tools are attached with the BHA, LWD can take measurements even in highly deviated wells. This is not always possible when using EWL.

2.1.3 Importance of Quality Control

Modern well logging techniques generate an abundance of data in order to accurately describe the geologic formations in a borehole. A rapid increase in the quantity of data created problems with data storage. Traditional data storage solutions proved to be too inefficient, resulting in petrophysicists spending a lot of time searching for data. The raw drilling data was stored

in different data formats, and would often be of poor quality.

These issues sparked the idea of a National Data Repository (NDR) for exploration and production related data. In 1995 the Norwegian Petroleum Directorate (NPD) initiated the Diskos NDR in an attempt to solve these issues. With the introduction of the Diskos NDR, NPD also published a set of guidelines for reporting petroleum data to the authorities. This set of guidelines is known as the *Blue Book* and is available at NPD's websites [6]. The Blue Book specifies the content, quality, format and structure requirements for various types of data. Raw well log seismic data, well composite logs and petrophysical interpretations are a few types of data collected by the NPD. All operators are obligated by law to report data from the Norwegian continental shelf (NCS) to the NPD [7]. These regulations provided a storage solution where high quality data is organized by a predefined standard. Companies like Logtek AS perform quality control and organize wellbore data before it is reported to the NPD. Logtek mentions that operators can benefit from this internally as well:

> However, oil companies have seen the added value and competitive advantage of organized and quality checked data for internal use, and for these reasons have implemented the same reporting routines for old and international well data [8].

The improved organization and data quality allow for easier machine learning integration and big data analysis. Data acquisition for training and test datasets is not as daunting because of a predefined format. High quality petroleum data implies that fewer decisions must be made during preprocessing of a dataset, resulting in fewer errors and bad choices.

2.2 Related Works

A possible use case for big data and deep learning in the petroleum industry is estimation of missing data. In [9], Onalo et al. present a data driven approach to well log predictions. Their approach is based on an artificial neural network with a single hidden layer. The proposed model utilize existing data of gamma ray logs, bulk density logs and shale volume to predict

or estimate the travel time of compressional and shear waves in a lithology. The results indicate the proposed model is a reliable and robust solution to estimating sonic transit time. The authors suggest their approach can be used to estimate missing data from old well logs, but it can also be used as a cheaper alternative to sonic well logging.

The existence of big data in the petroleum industry is not only contained to well log data. In [10], Sagheer and Kotb introduce a deep learning approach to time series forecasting of petroleum production. The authors propose a model based on a stacked long short-term memory (LSTM) architecture. The input to the model is a time series of water injection and production quantity from two different oil fields. The model is configured in two different scenarios: static and dynamic. The static configuration make predictions using only the actual observations in the training dataset, while the dynamic configuration is updated to use the output of previous predictions as well. Their results show that the deep LSTM architecture outperforms other deep recurrent neural networks and deep gated recurrent unit networks. The authors state the proposed approach was specifically tested on time series of petroleum production, but it can be applied to the majority of forecasting problems.

Borehole imaging is a special type of well log data where the features of an observation is recorded in multiple dimensions. The purpose of this process is to create an image of the wellbore walls at each depth interval, effectively creating a time series of images. In [11], Valentín et al. utilize borehole images to automatically identify the lithologies of the wellbore. The proposed approach is a deep residual convolutional network which uses blocks of 100 inputs from microresistivity and ultrasonic borehole image logs. The authors take advantage of the concept of residual blocks proposed by He et al. [12] to improve the accuracy and reliability of the model. The result of this study shows the model is able to extract more information from borehole image data compared to others methods.

It is evident that deep learning has the potential to improve current solutions in the petroleum industry. The three approaches proposed above focus on feature estimation, production prediction and lithology classification. Another common use case for deep learning on time series is anomaly detection. Being able to identify anomalous or erroneous observations has an abundance of use cases in the petroleum and non-petroleum industry.

In [13], Liu et al. propose a model which utilize the combination of a Attention Mechanism-based Convolutional Neural Network (AMCNN) and LSTM. The proposed model attempts to identify anomalies in edge devices in the Industrial Internet of Things (IIoT). The first component of the model, AMCNN, prevents gradient dispersion problems and memory loss by identifying the most important features. The second components, LSTM, is used due to its advantages on time series prediction. The proposed model is able to accurately detect anomalies, while also reducing the communication overhead. The study also provides an example of how the deep learning model can be trained in real time, which could be applied to real time wellbore drilling and analysis.

An interesting example of deep learning is the anomaly detection concept delayed LSTM proposed by May et al. [14]. Their approach utilize multiple LSTM-based models with delayed prediction to accurately identify anomalies in a time series. The training input to their model is a uni-variate time series of only non-anomalous data points. A separate observed dataset is assumed to contain a single anomaly, with all consecutive points being abnormal. The authors utilize multiple LSTM-based model to help reduce the impact of different types of noise on the time series. The input to the model is a window of n observations which are used to predicted the next n observations. The model is configured to generate output from 2 or 10 different LSTM networks. When the model obtains the actual values of the predicted area, the actual values are compared to all the different predicted values. The model selects the best predicted value for each observation in the window. The prediction of each individual observation in the window may come from different internal LSTM models. This concept effectively delays the output prediction of a window until the actual values are obtained. Their results show that this approach more accurately identifies anomalies than other methods, like stacked autoencoders and variational autoencoders, on both fictitious and real datasets.

In [15], Martí et al. introduce an anomaly detection approach for petroleum industry applications. The proposed approach use a combination of yet another segmentation algorithm (YNSA) and one-class support vector machine to detect anomalies in turbomachines. The segmentation algorithm in responsible for detecting sections of similar data across the entire time series. All the sections are fed to a one-class SVM which learns what is considered normal behavior. Any sections not conforming to this standard is

flagged as an anomalous section. The authors state the proposed approach was compared to real life applications in order to understand the validity, viability and performance of the approach. The results of this study shows the approach is able to outperform an existing automatic supervision system in a Brazilian petroleum company. This study is a perfect example of how deep learning technology can used to improve existing systems.

Chapter 3

Methodology

This chapter introduce the theoretical background for the methodology in this project. First, we present the file types used for well log data, and a detailed description of the features. Second, we present some important steps in the preprocessing pipeline, after which we start introducing some machine learning fundamentals. Next, we provide a detailed description of the theory behind our deep learning models. Finally, we discuss what evaluation metrics we use, and briefly mention some important libraries.

3.1 Dataset

The amount of well log related research is quite limited despite the existence of an abundance of historical and recent well log data. This is partly due to two issues associated with well log data: availability and old file formats. Well log data is considered market sensitive information and is typically not publicly available for researchers. The small amount of available well log data will often suffer from problems related to quality and quantity. Another big issue is the use of old file formats to store the data. These file types will often require the use of complex or expensive tools to extract the data.

3.1.1 File types

The majority of the well log data is generated, exchanged and stored in old and outdated file formats despite the recent explosion in technological advancements. The file type of choice may change depending on the oil or service company responsible for recording the well log. The following file types are the three most common file formats used for composite logs.

LIS

Log Interchange Standard (LIS) is a well logging format based on the VAX binary information standard. The format was developed by Schlumberger in 1974 and is known to be very difficult to work with [16]. LIS files are typically associated with an immense volume of historical data, but also see some use in Mud and Composite logs today. A single LIS file can contain one or more logical LIS files. Each logical file contains three key components: meta-data, index curve and measurement curves. The meta-data is stored as a set of records of different types. The index curve can be either depth or time based, and defines a consistent interval of measurements. Each measurement curve may be either single- or multi-dimensional, and contain one or more samples per depth/time interval. This means each measurement curve can record several values from multiple angles at each depth index.

DLIS

Digital Log Interchange Standard (DLIS), formally know as API RP66, is the successor of LIS and is currently the most common digital well log format. The standard was first introduced by the American Petroleum Institute in 1991 as a part of the Recommended Practice 66 for storage and exchange of well log data [17]. The format exists in two different versions, V1 and V2, due to the introduction of a revised version in 1996. V1 became the standard format after V2 failed to gain any traction. Like LIS, a DLIS file also contains one or more logical files. Each logical file contains their own DLIS sets and frames. A DLIS set typically contains the metadata related to the logging run. The DLIS set can be viewed as a table of information about all available DLIS frames, and what tools and parameters were used during

logging and processing of the data. The data recorded during logging is stored in multiple DLIS channels. A DLIS channel stores data as a function of depth or time, and is typically one-dimensional. The format also supports multi-dimensional DLIS channels, where multiple samples or multiple angles can be recorded for each step. A DLIS frame is simply a collection of multiple DLIS channels with the same depth or time axis. Despite the wide spread use of the standard, it is evident that this binary data format is very old. The format was introduced during a time where saving disk space played a key role to the success of a standard. An abundance of non-standard data-types and complex data structures were implemented in order to save disk-space. The lack of easily accessible programming tools, and the existence of company-specific dialects has recently sparked an interest in new formats such as JSON Well Logging Format [18].

LAS

Log ASCII Standard (LAS) is a file format published by the Canadian Well Logging Society around 1990 [19]. Unlike LIS and DLIS, LAS files are not binary and store all the information in human readable ASCII text. This means that researchers and developers can avoid using complex or expensive software to handle these files. The combination of a simple syntax and non-binary data is the main reasons why the standard is still popular today. One drawback of LAS files is that they require a lot more storage space than DLIS and LIS files with the same data. Consequently, LAS is not suitable for large volumes of well log data. Another drawback of the format is that the simple and ambiguous format description has caused an emergence of multiple dialects and semantic interpretations. The different dialects combined with the existence of three differentLAS versions (1.2, 2.0, 3.0) will often be an issue for researchers and developers.

3.1.2 Feature description

As stated in section 2.1.2, a composite log is generated by splicing together different types of well logs. What features are included in the composite log may vary depending on its purpose. However, the following features are usually found in the majority of composite logs and are regularly used in

petrophysical analysis:

- Measured Depth (DEPTH): This feature defines the index curve of a well log. The feature is typically measured in inches/10, and measures the length of the wellbore along the well path. It should not be confused with true vertical depth which measures the vertical distance from the surface.

- Acoustic Compression (AC) & Acoustic Shear (ACS): These feature are associated with sonic logging, and measures how fast elastic seismic compressional and shear waves travel through a formation. The features are mainly used to calibrate and support seismic data, and calculate formation porosity.

- Bit Size (BS): The bit size feature defines the diameter length of the drill bit at the current depth. The bit size of a wellbore will stay constant for sections at a time. The wellbore operator will first drill a hole with a specific bit size before inserting a slightly smaller casing. After inserting the casing, a smaller bit is attached to the drill in order to continue drilling.

- Caliper (CALI): This feature measures the variation in the wellbore diameter. The measurement is used as an indicator to detect cave-ins and shale swelling along the wellbore path. It is important to detect these types of situations as data from other well logs will be affected.

- Density (DEN): This feature measures the bulk density of a formation. Density, sonic and neutron porosity are the three logs that are used to calculate a formations porosity.

- Density Correction (DENC): This feature is used to correct and give more context to the density feature. This is necessary due to bulk density not being an intrinsic property, meaning it can change based on a variety of factors. The density is typically corrected based on pressure and temperature.

- Gamma Ray (GR): This feature measures the gamma radiation that naturally occurs in sedimentary rocks. The main use of this feature is to identify and differentiate different types of rocks. This is possible due to different sedimentary rocks emitting different levels of natural

gamma radiation. Shales will typically have a higher gamma radiation than other rocks due to the radioactive potassium found in its clay minerals. This feature can spot a clear difference between shale and non-shale rocks, but it struggles to differentiate sandstone and other carbonates due to the similar radiation levels. To overcome this problem, gamma ray logs and stratigraphic logs are analyzed together to properly identify the sedimentary rocks.

- Neutron Porosity (NEU): This feature measures the hydrogen index of a material. The feature is recorded using a neutron source to measure the concentration of hydrogen atoms. The main use of this feature is to estimate the amount of liquid-filled porosity, which is important for quantifying oil and gas reserves.

- Photoelectric factor (PEF): This feature measures a formations absorption of low-energy gamma rays. This feature is far less sensitive to differences in pore volume compared to NEU and DEN logs. PEF is a more detailed indicator of mineralogy and works very well on thin layers of sedimentary rock. This feature is also very useful in conjunction with DEN and NEU logs to identify mixtures of minerals in complex carbonates.

- Medium Resistivity (RMED) & Deep Resistivity (RDEP): These feature are used to measure the resistivity of a formation. Resistivity is measured at different distances away from the borehole and is recorded in ohm meters. The purpose of these features is to give information about the water saturation, formation porosity and the existence of hydrocarbons. The resistivity readings can also be used to differentiate between shale and non-shale rocks.

3.1.3 Aliases

Each feature in a well log is called a log curve and typically has a long descriptive name. These names are too long to print on well log headers and is therefore replaced with an alias. Some aliases are simple abbreviations, like GR for gamma ray, while others are mnemonics which are much harder to understand. Mnemonics are designed to be short to save storage space, and they are a mix of standard and vendor specific names. In the petrophysical handbook by E.R. Crain [20], more than 1500 aliases are listed for only 13

features. He also states that his list is far from complete due to new ones being constantly generated by service and log digitizing companies. Table 3.1 shows some of the known aliases for the available features in our dataset.

Feature	Aliases
DEPTH	Depth, DEPT
AC	DT24, DTC, HDT
ACS	DTS, DT4S
BS	HBS
CALI	CAL, HCAL, HCAL_1, HCALI, RSO8
DEN	HDEN, HRHO, RHO8
DENC	HCOR, HDRH
GR	HGR, EHGR, HDRHO, HNPHI, HRHOB
NEU	HCN, HNPO, HPHI
PEF	PE, HPEF
RDEP	HDR, HRLD
RMED	HRM, HRLS

Table 3.1: Subset of known feature alises.

3.2 Preprocessing

3.2.1 Missing Values

One of the first preprocessing steps is to decide what approach is most suitable to deal with missing data in the dataset. Handling missing data is a critical step as the majority of machine learning models will not function otherwise. It is not possible to define a single optimal approach for handling missing data as it is heavily reliant on the available dataset and the machine learning algorithm of choice.

It is also important to understand why the data is missing in the first place. Traditionally there are three categories of missing data mechanisms: Missing at Random (MAR), Missing Completely at Random (MCAR), and Not Missing at Random (NMAR) [21, 22]. MAR is a mechanism where the missing values are randomly distributed across a subset of the data. The missing values are related to the observations, and not related to the

features. MCAR is the most strict mechanism and is only present if the missing data is randomly distributed across the entire dataset. MCAR can be viewed as a special case of MAR where the distribution of missing data is independent of the observations and features [22]. The final mechanism NMAR applies when the missing data has a structure to it. This mechanism can be challenging to work with as the only solution is to create a model that accounts for missing data, and use it to develop an unbiased estimate. However, in some cases where the missingness cannot be accounted for, the introduced bias can be negligible [23].

The removal of observations or features with missing values is the simplest approach to handling missing data. The benefit of removing missing data is a complete dataset without any outside influence on the data. However, the removal of observations and features might lead to loss of valuable information, and introduce a bias towards the remaining observations. If the MCAR mechanism is satisfied, the removal of observations will not introduce a bias [24]. However, a bias might be introduced if the MCAR mechanism cannot be satisfied. It is heavily debated whether or not the proportion of missing values should be taken into consideration when removing missing data. Some research suggests that if the proportion of missing data is above 40%, the amount of missing data is so substantial that only the observed data should be included [25]. Additionally, if the dataset contains less than 5% missing values, the proportion of missing data is so negligible that it can safely be removed from the dataset. However, some researchers disagree with this statement and suggest the proportion of missing data should not be used as an indicator for the removal of observations or features [26].

Another common method to deal with missing values is with the use of imputation. The most simple version of imputation is single imputation, where each missing value is replaced by an estimated value of the observed data. There are a variety of different single imputation methods available to estimate the missing values. The most common single imputation method is to calculate and replace all missing values with the mean, median or mode of the appropriate feature. This method can work well if the proportion of missing data is small. However, as the proportion of missing data increases, single imputation will underestimate the variance and introduce a bias in the data. This problem is not dependent on the missing data mechanism and should only be used with great caution [25, 27]. Other time series specific simple imputation methods like Last Observation Carried Forward (LOCF),

Next Observation Carried Backward (NOCB) and linear interpolation face similar issues. These methods attempt to replace missing sections of data with the last valid observation, next valid observation or a set of linear values between the two.

One of the problems with single imputation is that the imputed values are treated equal to the observed data, which often cause a misleading analysis. Multiple imputation attempts to solve this problem by replacing a missing value with a set of possible values. The values are generated and chosen by a defined imputation model [25]. These sets of values are used to create multiple candidate datasets which are analyzed individually using standard analytical procedures. The final step involves using the result of all the candidate datasets and combine them into a single multi-imputation result. This approach has been proven to retain the natural variability of the missing values, which provides a valid statistical inference [22, 25, 23].

3.2.2 Feature Scaling

Feature scaling is an essential step in data preprocessing and involves transforming each feature in the dataset to be on the same scale. When the features in the training dataset are on different scales, the machine learning algorithms tends to favor features with larger range [28]. Neural networks in particular will heavily adjust the weights of features with larger scales, while the weights adjustment of features with smaller scales are minuscule in comparison. This in turn will slow down the learning rate and convergence of the network, and prevent the algorithm to effectively learn from the dataset.

Standardization, also called Z-score normalization, is a method of scaling features to ensure a mean $\bar{x}$ of 0 and a standard deviation σ equal to 1. The scaled value x' is calculated by subtracting the mean from the original value x before dividing the result on the standard deviation. Unlike other methods, this feature scaling technique is not bound to a range. Standardization is most useful if the dataset has a Gaussian distribution, but it can be used on data with other statistical distributions. One of the perks with standardization is that it is much less affected by outliers compared to other normalization techniques.

$$x' = \frac{x - \bar{x}}{\sigma} \tag{3.1}$$

Min-max normalization, also called min-max scaling, is a method of scaling features to the range [0, 1]. The scaled value x' is calculated by subtracting the minimum value x_{min} from the original value x before dividing the result on the difference between the maximum x_{max} and minimum x_{min} values. This formula ensures the minimum scaled value will be equal to 0 and the maximum scaled value will be equal to 1. This feature scaling technique preserves the relative distance between values and is useful in situations where the distribution of the data is unknown. One of the issues with min-max normalization is that it is very sensitive to outliers. The value of the outliers at both ends of the spectrum are directly used in the formula, which has a significant impact on the scaled values. A single large outlier can scale the majority of the values in the feature between 0 and 0.1, while the outlier would be scaled to 1. In this situation a machine learning algorithm would struggle to learn from the data as there are no values in the range (0.1, 1), and the difference between normal values are negligible.

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.2}$$

3.3 Fundamentals & Layers

3.3.1 Supervised and Unsupervised

Machine Learning (ML) is a form of artificial intelligence that employ statistics to learn from an input dataset. An ML algorithm is usually trained using supervised or unsupervised learning. During training, a supervised ML algorithm is given the target labels. In classification tasks, the algorithm attempts to classify a given input as one of two or more labels. The algorithm is penalized for wrong classifications and is thus encouraged to change parameters in order to improve future predictions. In regression tasks, the algorithm instead predicts actual continuous values. The ML model is penalized based on the similarity or closeness between the prediction and corresponding target label. After sufficient training, supervised

models can make predictions on new data of the same type as the training data. Unsupervised ML algorithms are used when there are no available target labels. This means that unsupervised cannot be trained to solve neither classification nor regression tasks. Instead, features and the relation and similarity between input entries are learned. Unsupervised ML techniques group input entries into various clusters. This is called clustering and can be used to categorize input entries without having prior knowledge of the dataset.

In this project, we are originally dealing with an unsupervised ML task as there are no available labels. However, the algorithms that we use either tries to compress and decompress to make the output look like the input, or predict the next input. This means that the output is on the same format as the original dataset. Thus, the target labels is the input itself. The algorithms are therefore capable to supervise themselves. We can therefore say that the algorithms are self-supervised. Furthermore, this is also a regression task as the input consists of features of continuous values.

3.3.2 Activation Functions

An activation function is a function that applies a non-linear (usually) transformation on the input. The transformation is applied element wise if the input is not a scalar. In artificial neural networks, activation functions are often applied to introduce non-linearity in an otherwise linear system.

Sigmoid

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.3}$$

The sigmoid function (3.3) transforms the input to a value between 0 and 1. This activation function is mainly used to ensure that the output of a neural network is within well defined bounds.

Hyperbolic Tangent

$$f(x) = \tanh(x) = \frac{e^x - e^{-1}}{e^x + e^{-1}} \tag{3.4}$$

The hyperbolic tangent function (3.4) is similar to the sigmoid function, except the output range from -1 to 1.

Rectified Linear Unit

$$f(x) = \max\{0, x\} \tag{3.5}$$

The Rectified Linear Unit (ReLU) activation function (3.5) transforms all negative values to zero. Positive terms remain unaffected.

3.3.3 Deep Learning Basics

A Feed-Forward Neural Network (FFNN) is the most basic form of an artificial neural network. The name of the network is derived from the direction of the information flow in the network. The information moves from the input layer, through one or more fully connected intermediate layers and to the output layer. Each intermediate layer contains a set of neurons that feed a weighted input through a linear or non-linear activation function to generate an output. The network use an error function and a back-propagation technique to adjust the weights of the input connections. Then the network computes the error between the output and target values. The algorithm back-propagates the error through the network, adjusting all the weighted connections to minimize the value of the error function. The value of the error function is only reduced by a minor amount for each iteration. This process is repeated multiple time until the network converge to a state where each new update to the weights provide a negligible change in the error value. A FFNN typically utilize gradient descent, a non-linear optimization technique, to optimize the process of adjusting the weighted connections.

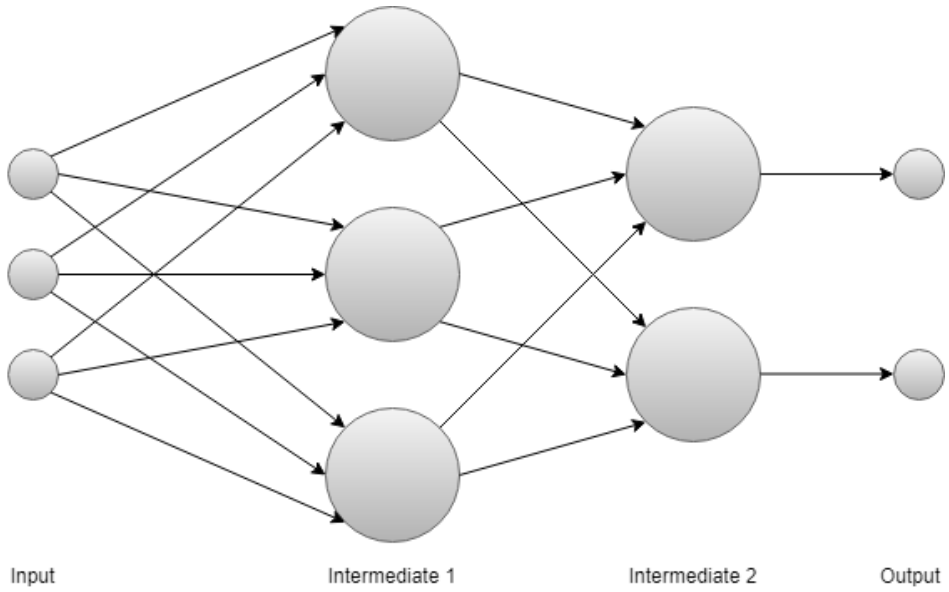


Figure 3.1: Basic feed-forward neural network architecture.

3.3.4 Recurrent Layer

One of the major problems with FFNN and time series data is that it has no memory of previous data. This is due to FFNNs only using the current input during training and prediction. This means the network has no notion of time and struggles to predict whats coming next. An Recurrent Neural Network (RNN) attempts to solve this problem by expanding on the FFNN structure by introducing a loop in each neuron. Each neuron will now store its output as a hidden state to be used in the next iteration. The hidden state of the neurons acts as the networks short-term memory of previous data. In an RNN, each neuron in the intermediate layer concatenate its weighted inputs and hidden state and feed it to the activation function.

After feeding the data through the network and calculating the output, the network has to back-propagate the error through the network to adjust the weighted connection. Like FFNN, all neurons that took part in the calculation of the output should have its weights updated. The key difference is that the output h_t is dependent on all the weights used to calculate the hidden state for all previous timesteps. The network uses the weighted inputs x_t and the previous hidden state h_{t-1} to calculate the output h_t. This

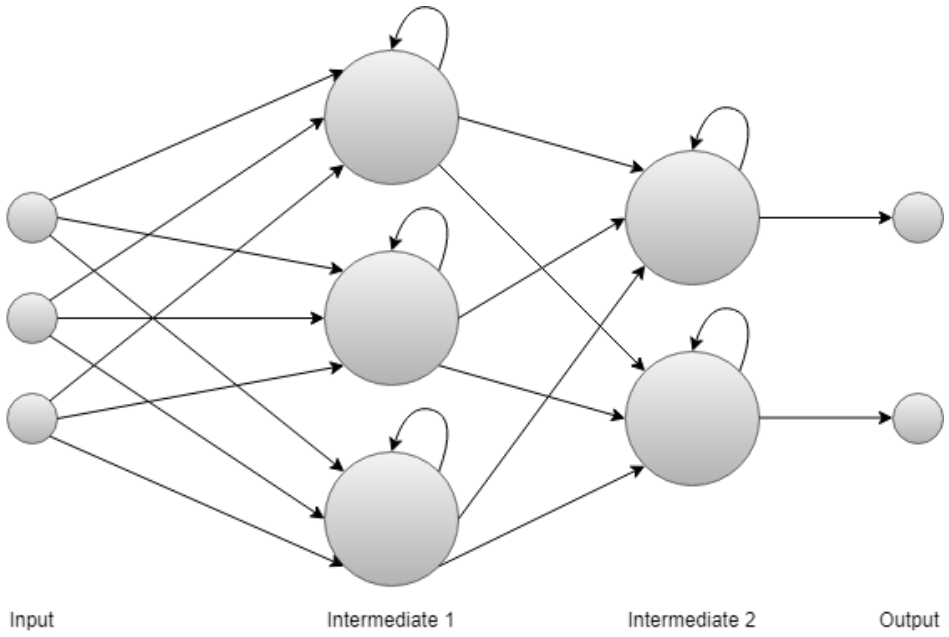


Figure 3.2: Recurrent neural network architecture.

means the network has to back-propagate the error from the last timestep all the way back to the first timestep in order to adjust all the weights. This technique is called Back-Propagation Through Time (BPTT), and when combined with gradient-based learning methods is the root of the vanishing gradients problem in RNN [29]. The hidden state, like with all other connections in the network, is associated with a weight. These weights are typically assigned with a random value close to zero at the beginning of the network. The hidden state will be multiplied with the same weight multiple times when the network moves from one timestep to the next. The repeated multiplication of a number close to zero means that the gradient becomes smaller and smaller for each timestep. When a network has a low gradient the network stop learning due to insignificant weight adjustments in each timestep. The opposite problem, exploding gradients, occurs if the network has a very high gradient. The weights in this scenario would receive huge adjustments in each timestep, causing the network to be unable to learn from the training data.

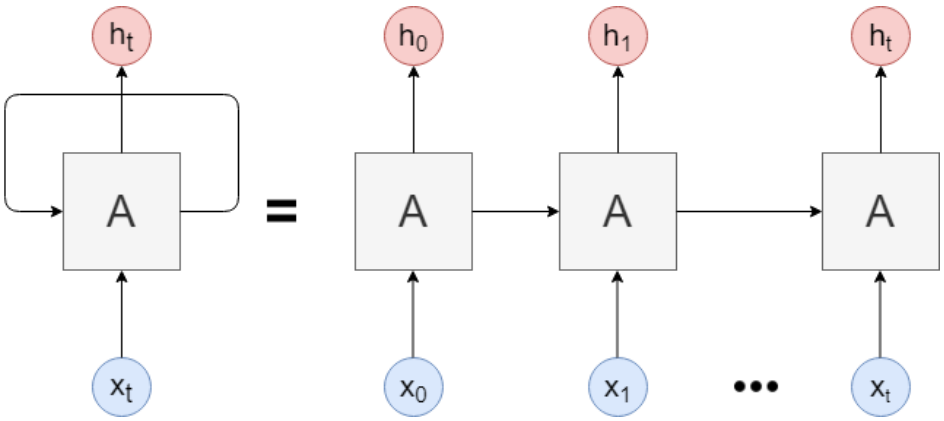


Figure 3.3: Unrolled representation of an RNN neuron.

3.3.5 Convolutional Layer

A convolutional layer uses a set of filters to extract the location of various features from its input. In a Convolutional Neural Network (CNN), each filter in a convolutional layer is optimized to learn a single feature. The output of the layer is obtained by convolving the filter over the input. In 2D convolutions, the output is called a feature map. In a CNN with multiple convolutional layers, the deeper layers generally extract more complex features compared to the previous layers [30]. Exactly what these features are is learned by the network during training. In an image classification CNN, feature maps of early layers may only contain features such as edges, while subsequent layers may pick up on eyes, noses, ears etc. The dimensions of the convolutional layer change depending on the input data and use case. Some examples of CNN applications include time series (1D), images (2D), video (3D), and VR (4D). In our use case the input is a multivariate time series, which means the CNN use a multivariate 1D convolutional layer. Furthermore, the mathematical representation shown in this section aim to illustrate how a kernel convolves over a multivariate time series input. Thus, this representation is not optimized like in an actual implementation.

The input space of a multivariate 1D convolutional layer is (n, C_{in}) where n is the length of a sequence of records and C is the number of channels. A channel is typically a feature in a dataset. The input space is denoted by the variable X. A kernel has the shape (k, C_{in}) where k is the kernel

size. A layer has C_{out} kernels where the j-th kernel is denoted by K_j. The convolutional layer computes its output by convolving each kernel along the input space's temporal axis. To simplify the notation, a row in X and K_j is denoted as x_i and w_p respectively. Both x_i and w_p are vectors of size C_{in}. The subscript of w is centered around zero where $p \in [-P, P], \quad P = \lfloor \frac{k}{2} \rfloor$.

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{i-1} \\ x_i \\ x_{i+1} \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix}, \quad K_j = \begin{bmatrix} w_{-p} \\ w_{-p+1} \\ \vdots \\ w_{-1} \\ w_0 \\ w_1 \\ \vdots \\ w_{p-1} \\ w_p \end{bmatrix} \tag{3.6}$$

A convolution is computed by sliding the kernel over the input and computing the element-wise product sum of the input and the overlapping kernel. Assuming an odd kernel size, the center row of the kernel is lined up with x_i. Lets say that $k = 3$, the convolved output of x_i would be the sum of the dot products between the overlapping rows of the input and the kernel. The output for the i-th row convolved with the j-th kernel becomes: $x_{i-1} \cdot w_{-1} + x_i \cdot w_0 + x_{i+1} \cdot w_1$. Since the kernel is centered around x_i, this implies that the kernel extends beyond the input when $i < P$ or $i > n - P$. For these values of i the convolution cannot be computed, meaning that the output shape for a single kernel convolved across the entire input space is $(n - 2P, 1)$. For multiple kernels, the output shape becomes $(n - 2P, C_{out})$ where the j-th column is the output of the j-th kernel convolved across the input. Equation (3.7) shows how a single value in the output space is computed. This formula assumes that the stride length and dilation rate are both set to 1. Strides and dilation rate are explained in later paragraphs.

$$Y_{i,j} = \sum_{p=-P}^{P} x_{i+P-p} \cdot w_p, \quad w_p \in K_j \tag{3.7}$$

If it is important that the output has n rows, the input can be padded by P rows at the beginning and end. Enforcing that the output has n rows is typically referred to as *same* padding. Choosing the values of the padded rows usually requires knowledge of the context of the input, as it can have a significant impact on the output. If the input channels are cyclical it might be a good idea to set the leading padded row to equal the last P rows, and the trailing padded rows to equal the first P rows. Other options are to use the mean of the input channels or simply use zeros. Zero-padding won't affect the computations at the cost of not knowing the full context at the ends of the input.

Causal padding only adds padding at the start of an input sequence. This also modifies how the kernel overlaps with the input. The final kernel element becomes the reference point instead of the center kernel element. For $k = 3$, the output for the i-th row convolved with the j-th kernel now becomes: $x_{i-2} \cdot w_{-1} + x_{i-1} \cdot w_0 + x_i \cdot w_1$. This effectively prevent the kernel from learning from the future, and may therefore be more suitable for time series data. This is also referred to as causal convolutions [31].

Aside from padding, the stride length also affects the size of the output space. The stride length s affects how much the kernel slides over the input. Until now we have assumed an $s = 1$, meaning that the kernel convolves over every input row. A stride of length 2 means that the kernel only convolves over every other input row. The benefit of increasing the stride length is that it decreases the amount of computations by a factor of s. The input may become degraded when choosing an s too large.

Additionally, the dilation rate also affects the output space. Dilation rate d defines the spacing between the kernel elements w_p, increasing the kernel size. This effectively adds $d-1$ new dummy elements between each original kernel elements. Each new element contains only zeros, meaning that the new elements don't directly affect the output because the dot product of itself and another vector is zero. It is therefore not strictly necessary to expand the kernel as it is unnecessary to perform dot products that always yields a value of zero. Instead it is better to manipulate the index. Thus far we have assumed a dilation rate $d = 1$, meaning that the kernel element has no additional spacing. When $d = 2$, w_0 is still centered at x_i but w_p now overlaps with x_{i+dp} instead of x_{i+p}. Without additional padding the output will have a shape of $(n - 2dp, C_{out})$. Increasing the dilation rate is

useful when it is desired to expand the context in the convolution without increasing the actual kernel size. Increasing dilation rate too much may provide too much temporal distance between the datapoints involved in the convolution. One example, in finances, each row in the input contain the daily changes in stock value for multiple companies (channels). Now let's say that $d = 7$. This means that the kernel only overlaps with rows that corresponds with the same day of the week. As a result, the convolution will not be able to discover any hidden context or patterns for day to day changes in stock value, only patterns for Monday to Monday and Tuesday to Tuesday etc.

$$n_{out} = \left\lfloor \frac{n + 2 \times padding - d \times (k-1) - 1}{s} + 1 \right\rfloor \tag{3.8}$$

As previously described, carefully choosing the stride length, dilation rate and the padding mode can improve the filter in extracting hidden patterns or features from the input. Additionally, increasing either the stride length or the dilation rate can significantly reduce the number of computations necessary for obtaining the output. These parameters also affects how many rows n_{out} there are in the output. Thus the output shape of a 1D multivariate convolutional layer is (n_{out}, C_{out}).

3.3.6 Pooling Layer

The pooling layer downsamples the input space. Like the convolutional layer, the pooling layer performs the downsampling by sliding a kernel along the input's axes. Applying padding, strides and dilation in a pooling layer is identical to applying those concepts in a convolutional layer. Unlike the convolutional layer, the pooling layer's kernel has no weights and only operates on one channel at a time. Thus, the kernel does not overlap with multiple channels. The kernel performs a function on the input elements it overlaps with. This function typically finds the max or the average value of the overlapping elements and are usually referred to as Max Pooling and Average Pooling respectively.

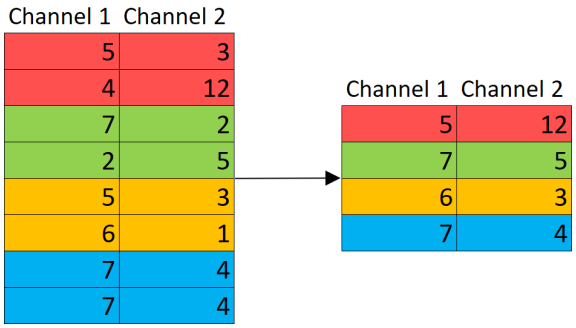


Figure 3.4: Max pooling example with kernel size & stride length of 2 and dilation rate of 1.

Figure 3.4 shows the output of a 1D Max Pooling layer in relation to a input of shape (n, C). The kernel has the shape $(k, 1)$ and outputs only the maximum value it overlaps with for every iteration and for every channel independently. The output shape becomes (n_{out}, C) where n_{out} is conveniently the same as in a 1D convolutional layer (defined in equation 3.8).

3.3.7 Dropout Layer

Random dropout is a regularization technique to prevent overfitting. This layer has one hyperparameter p which specifies the fraction of neurons that are disabled for a training pass. The neurons that are disabled are chosen at random. With dropout, neurons becomes better at detecting useful features [32].

3.4 Algorithms

3.4.1 Autoencoder

An Autoencoder (AE) extracts the most important features of an input. The AE then tries to reconstruct the input based on the extracted features. This is therefore a self-supervised algorithm since the input is also used as the target. The term *autoencoder* generally refers to the structure of

the model illustrated in figure 3.5. The purpose of the hourglass shape is to filter out irrelevant features. The autoencoder consists of two separate neural networks: the encoder and the decoder. The encoder can be used separately to perform dimensional reduction on the input. The autoencoder is also the technology behind what is known as deepfakes which allows for animating one's facial expressions onto another person's face. This can be done by training one encoder that detects facial features and two decoders, one for person A and one for person B. By encoding person A's face and decoding using person B's decoder, it will seem like person A's facial features are projected onto person B's face [33]. In this project we use AE for anomaly detection. This is based on the assumption that erroneous and trend deviating observations are reconstructed with higher error. A reconstruction error greater than a specified threshold indicates an anomaly. For this purpose, it is not necessary to use the encoder and decoder separately.

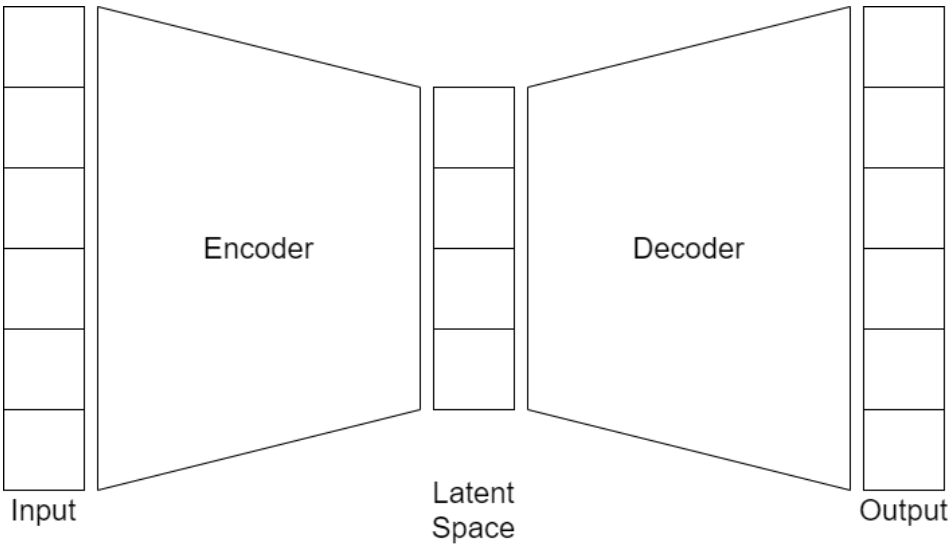


Figure 3.5: Autoencoder network architecture.

The encoder performs a dimensional reduction on the input features, similarly to a principal component analysis (PCA). Thus the output of the encoder must have fewer dimensions than the input. This output vector is coordinates in what is called a latent space. While PCA is good at finding the optimal linear subspace, it is limited with orthogonality constraints. An encoder also tries to find an optimal subspace, but it does not have to be

orthogonal with the input vector. Additionally, an optimized encoder can learn deep and non-linear contexts and project them onto a latent space with possible little information loss.

The decoder is trained to reverse any transformations done by the encoder. In other words, the decoder takes coordinates from the latent space and reconstructs data that looks like the original un-encoded vector. When combined into an autoencoder, the encoder and decoder works in tandem in finding the optimal latent space. An autoencoder may also be thought of as a lossy compression and decompression algorithm. The deep learning aspect helps minimize the loss.

3.4.2 Variational Autoencoder

An autoencoder attempts to encode and decode an input with as little reconstruction loss as possible, which often leads to overfitting. A Variational Autoencoder (VAE) introduce regularization techniques in order to avoid this problem. The regularization will also organize the latent space by introducing two properties: continuity and completeness [34]. The continuity propriety states that points close together in the latent space should give similar results once decoded. The completeness property states that if you sample and decode a point in the latent space, it should provide a meaningful outcome.

A traditional autoencoder satisfies neither of these properties. Similar features encoded with an autoencoder are not necessarily close to one another in the latent space. Additionally, there may be a void or emptiness between features in the latent space. This means that the decoder is unable to recognize what features a point in the empty space has. The decoded result of such a point may bear no meaning at all. The variational autoencoder overcomes these issues such that the latent space can be used in a useful and meaningful way. For instance, sampling and decoding a point between two features in the latent space will produce data that looks real and bear resemblance to both chosen features in the latent space. Figure 3.6 shows how a latent space looks like with and without regularization. As you can see, with regularization we can sample a point that inherits from all features. Without regularization, we would get something that doesn't resemble any of the features.

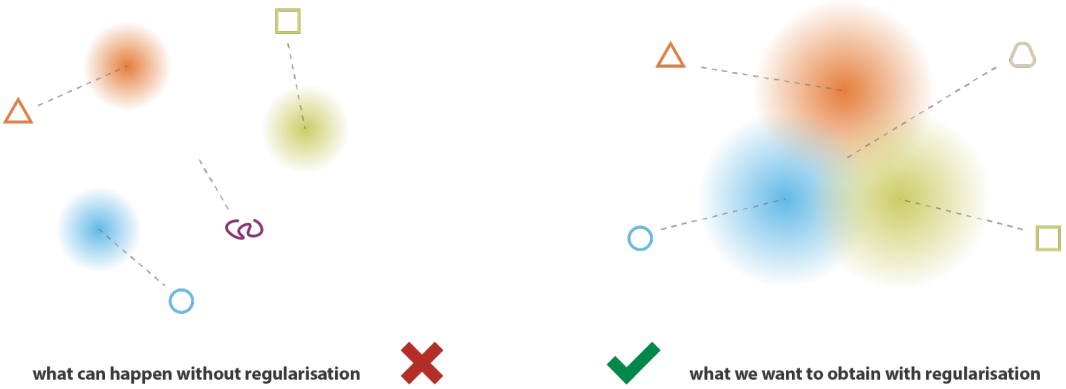


Figure 3.6: Latent space visualization with and without regularization. Source: [34].

The VAE introduce regularization by slightly modifying the architecture of an autoencoder. A VAE encode an input as a distribution over the latent space, rather than encoding an input as a single point. Typically the encoded distributions tend to be Gaussian distributions in order to use the mean and covariance matrix for training. The loss function used during training is a combination of a reconstruction term and a regularization term. The reconstruction term is typically Binary Cross-Entropy (BCE) loss or Mean Square Error (MSE) loss. Kullback-Leibler divergence is used as the regularization term to compare the difference between the returned distribution and a Gaussian distribution.

3.4.3 Long Short-Term Memory

Long Short-Term Memory (LSTM) networks were created to solve the vanishing gradient problem associated with RNN, and have shown to be well-suited in tasks related to time series data. An LSTM neuron is typically referenced to as a LSTM cell or unit, and is made up of a cell state and three gates. Figure 3.7 shows the structure of an LSTM cell at timestep t. LSTM cells enable RNNs to remember multiple inputs over a longer period of time due to the cell state. The cell state works as the core memory of the network, and transports information between timesteps. The cell is able

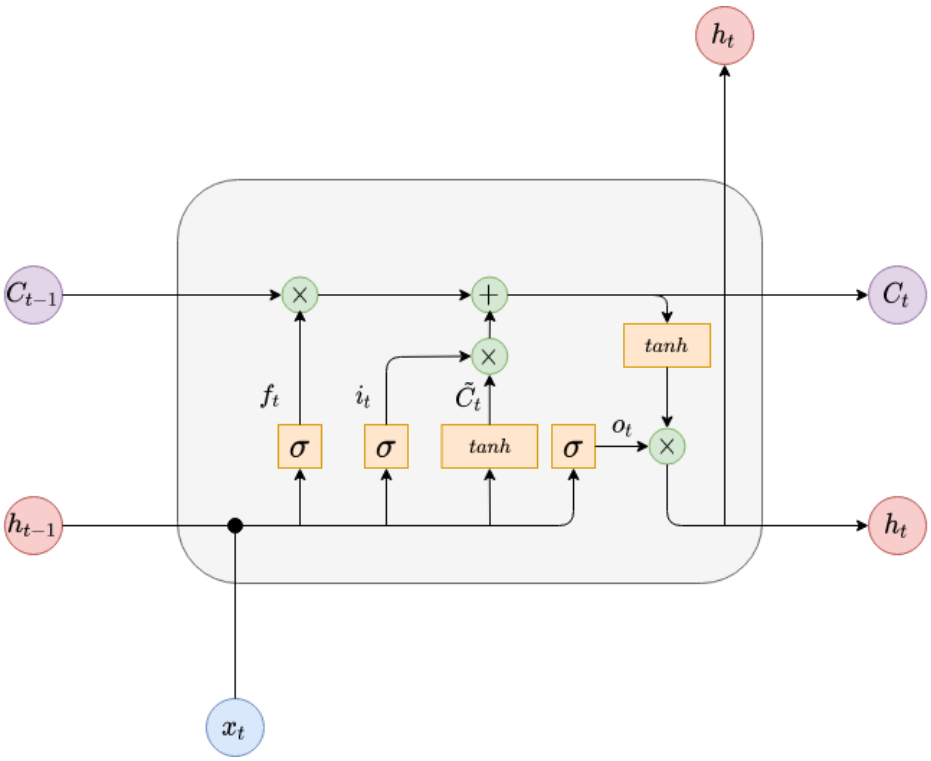


Figure 3.7: LSTM cell architecture.

to read, write and delete which information is relevant with the use of an input, forget and output gate.

The first step in the LSTM cell is the forget gate where the network identifies which information should be forgotten. The previous hidden state h_{t-1} and the current input x_t is combined using vector concatenation. This combined vector is then multiplied with the weight matrix W_f before adding the bias offset. This input is sent through a sigmoid function to determine which information should be forgotten. The sigmoid function will return a vector f_t with values between 0 and 1 which indicates the importance of each input. A value close to 0 indicates the input should be forgotten, while a value close to 1 should be kept.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.9}$$

The next step in the LSTM cell is the input gate, where the network identifies which information should be used to update the cell state. This gate takes two different inputs. The first input is a sigmoid layer that returns a vector i_t with values between 0 and 1. The purpose of this layer is to identify which input values should be updated in the cell state. The second input layer feeds the same information through a *tanh* function to create a vector $\tilde{C}_t$ of possible candidate values between -1 and 1 to be added to the cell state. The tanh function helps regulate the network and combat the vanishing gradient problem [35]. In this gate, $\tilde{C}_t$ defines the candidate values for the cell state update, while i_t scales the values according to how much we want to update the cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.10}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3.11}$$

After calculating the output of these two gates, the LSTM cell is ready to update cell state. First, the previous cell state C_{t-1} is pointwise multiplied with f_t to forget the values we want to forget. Second, the candidate values $\tilde{C}_t$ are pointwise multiplied with i_t to produce the scaled candidate values. Finally, pointwise addition is performed on the two vectors to generate the new cell state C_t.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.12}$$

The final step of of the LSTM cell is to compute the hidden state h_t output. The new hidden state contains information about the previous inputs, and is a filtered version of the cell state. The first step in this gate is to push the input values through a sigmoid layer to define what part of the cell state we want to use for the hidden state output. The calculation of the vector o_t is the same as the sigmoid layer in the forget/input gate but with the associated weights and bias. The cell state vector is fed through a tanh function and then pointwise multiplied with the output vector o_t to create the new hidden state. This multiplication will make sure the hidden state only contains the information we want from the cell state. Both the current hidden state and the cell state can now be sent over to the next timestep.

The hidden state of the final timestep is used as the output for the LSTM cell.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.13}$$

$$h_t = o_t * \tanh(C_t) \tag{3.14}$$

It is worth noting that the structure of an LSTM cell may differ from project to project. The original LSTM paper by Sepp Hochreiter and Jürgen Schmidhuber only included the LSTM cells, input gate and output gate [36]. The structure explained in this subsection is based on the architecture first proposed by Gerr et al. [37]. This network structure introduced a forget gate that enabled LSTM networks to reset its own state. Another variation of LSTM includes the implementation of "peephole" connections. Peephole connections were first introduced by Gers & Schmidhuber [38], and involved adding the previous cell state as input to the three sigmoid layers. The equation below shows how the forget gate equation would be updated if peephole connections were implemented.

$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f) \tag{3.15}$$

3.4.4 DeepAnT

In [39], Munir et al. propose a Deep learning approach for unsupervised Anomaly detection in Time series data (DeepAnT). The DeepAnT model utilize a deep convolutional neural network to predict the next instance in the time series. The prediction is based on a window of the previous n recorded instances. An anomaly score is calculated based on the Euclidean distance between the observed and predicted values. An instance is defined as anomalous if the anomaly score is greater than a specified threshold. DeepAnT appears to perform well on large streams of multivariate time series data, where normal and anomalous data is collected from heterogeneous sensors (much like sensors on a wellbore drill).

DeepAnT consists of two modules: a time series predictor and an anomaly detector. The time series predictor is a CNN consisting of two 1D convolu-

tional layers, each connected by a max pooling layer. The second pooling layer is followed by a fully connected layer. Both convolutional layers and the fully connected layer applies the ReLU activation function on their outputs. The final layer is the output layer which predicts the next (not yet observed) sequence of measurements.

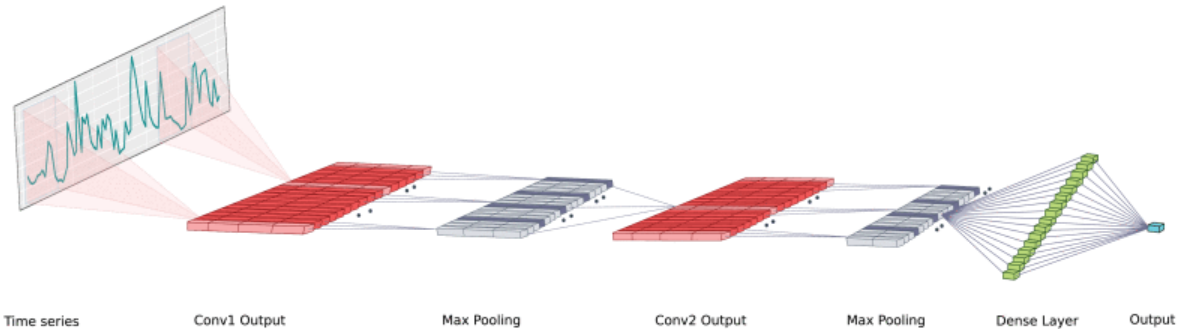


Figure 3.8: DeepAnT's network architecture. Source: [39].

DeepAnT uses Mean Absolute Error (MAE) as a loss function. MAE indicates the discrepancy between two components. It is preferred for its simplicity and when the two components expresses the same phenomenon and are on the same scale [40]. Here, the two components are the j-th observation y_j and prediction $\hat{y}_j$.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j| \tag{3.16}$$

The second module, the anomaly detector, computes an anomaly score by calculating the Euclidean distance (ED) between y and $\hat{y}$. The significance of an anomaly is based on the Euclidean distance. A high Euclidean distance means that y_t and $\hat{y}_t$ are vastly different. Assuming that the DeepAnT model generally makes good predictions, a high Euclidean distance implies y_t is anomalous because it is not similar enough to the expected value $\hat{y}_t$. A threshold needs to be defined in order to only label data anomalous if the Euclidean distance is larger than this threshold.

$$ED(y_t, y'_t) = \sqrt{(y_t - y'_t)^2} \tag{3.17}$$

"The proposed unsupervised approach incorporates context, seasonality, and trend into account for detecting anomalies." [39]. *Seasonality* can be regarded as non-abnormal fluctuations or change in expected data, similarly to when a drill enters a new lithology or lithostratigraphic unit. DeepAnT should be able to predict the next observed data based on the current trend. This is vital for our datasets as measured data from different lithostratigraphic units can be vastly different. Furthermore, DeepAnT expects that fewer than 5% of streaming data is erroneous. While our datasets are not directly streaming data, the composite logs are stitched together from streamed data and corrected throughout the quality control process. Consequently, our datasets should not contain a significant amount of erroneous or anomalous data. Based on these reasons, DeepAnT appears to be a well-suited model for our use case.

3.5 Model Evaluation

To efficiently evaluate the performance of a classifier, a confusion matrix is often used. The confusion matrix indicates to which degree a classifier correctly labels predictions in a precise and convenient manner. The rows constitutes the actual positive (P) and actual negative (N) instances in the dataset respectively, while the columns represent the predicted positive (PP) and predicted negative (PN) instances made by a classifier. Table 3.2 shows the representation of a confusion matrix. An instance correctly labeled as positive is defined as true positive (TP). Likewise, true negative (TN) is an instance correctly labeled as negative. An instance wrongly labeled as positive is referred to as false positive (FP). Similarly, an instance incorrectly labeled as negative is referred to as false negative (FN). FP and FN are equivalent with type I & II errors respectively.

	Predicted Positive (PP)	Predicted Negative (PN)
Actual Positive (P)	TP	FN
Actual Negative (N)	FP	TN

Table 3.2: Confusion matrix structure.

Looking at the confusion matrix may not yield immediate insight to a classifier's performance. To obtain additional insight, auxiliary performance measurements (metrics) are derived from the confusion matrix. The metrics that we use in this study are listed and explained below.

3.5.1 Accuracy (ACC)

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{3.18}$$

The accuracy measurement indicates the ratio of correct predictions versus all predictions. Accuracy is simple and intuitive to understand, however, these are also the reasons accuracy can be very misleading. The issue arises when there is an imbalanced ratio of P vs N, and TP or TN respectively is much larger than the other values in the confusion matrix. Both cases yields a high accuracy. The high accuracy tends to overshadow the presence of type I & II errors when TP or TN are large respectively [41].

3.5.2 Precision (PPV)

$$\mathrm{PPV} = \frac{\mathrm{TP}}{\mathrm{PP}} \tag{3.19}$$

Precision specifies the amount of TP among all PP, indicating how precise a classifier's positive predictions are. Thus, precision is also referred to as Positive Predicted Value (PPV). A high PPV (close to 1) implies a low presence of type I errors. Likewise, when PPV = 0, all positive predictions are type I errors because TP = 0 and PP = FP.

3.5.3 Recall (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{P}} \tag{3.20}$$

Recall indicates the ratio of how many instances where correctly labeled positive among all actual positive instances. Recall is also referred to as True Positive Rate (TPR), sensitivity and hit rate. TPR also shows the presence of type II errors. A TPR of 1 implies a total absence of type II errors.

3.5.4 F-score (F)

$$\begin{aligned} F_\beta &= (1+\beta^2) \cdot \frac{\mathrm{PPV} \cdot \mathrm{TPR}}{(\beta^2 \cdot \mathrm{PPV}) + \mathrm{TPR}} \\ &= \frac{(1+\beta^2) \cdot \mathrm{TP}}{(1+\beta^2) \cdot \mathrm{TP} + \beta^2 \cdot \mathrm{FN} + \mathrm{FP}} \end{aligned} \tag{3.21}$$

The F-score measures the performance of a classifier based on both precision and recall, where recall is β times more important than precision. F_1 is the harmonic mean between precision and recall and is the most commonly used F-score. A β value of 2 means that recall is twice as important than precision, and a β value of 0.5 means that precision is twice as important than recall.

3.5.5 Matthews Correlation Coefficient (MCC)

$$\mathrm{MCC} = \frac{\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN}}{\sqrt{\mathrm{PP} \cdot \mathrm{P} \cdot \mathrm{N} \cdot \mathrm{PN}}} \tag{3.22}$$

Precision, recall and F-score only focus on the positive case and neglects the importance of TN. The Matthews Correlation Coefficient (MCC) provides a balanced measure that indicate the classifiers efficacy of both positive and negative predictions. For instance, MCC is able to indicate that a classifier has poor performance even if the F-score indicates good performance. MCC is the only measure presented in this section that has a range of $[-1, 1]$ as opposed to $[0, 1]$. When MCC $= 1$ the classifier is perfect, while MCC $= -1$ means that the classifier make no correct predictions. A MCC value of 0 is equivalent to making random guesses. From (3.22) it is clear that MCC is undefined if one of the rows or columns in the confusion matrix is zero. Under these circumstances, we define MCC as Chicco et al. did in [41]. When one of P, N, PP or PN is zero we get MCC $= 0$. In the case only one of TP, FN, FP or TN is non-zero, MCC $= 1$ when TP $\neq 0$ or TN $\neq 0$. Conversely, we get MCC $= -1$ when FP $\neq 0$ or FN $\neq 0$.

3.5.6 Prevalence

$$\text{Prevalence} = \frac{\text{P}}{\text{P} + \text{N}} \tag{3.23}$$

Prevalence shows the ratio of positive instances versus all instances. A prevalence of 0.5 means that there are equal amounts of positive and negative instances, i.e. a perfectly balanced dataset.

3.6 Libraries

For reading and processing raw well logs we use the well log access library Log I/O [42]. We also utilize the unit of measure library UoM [43] to ensure the unit of a feature is the same for all well logs. These libraries are developed by Petroware AS, a daughter company of Logtek AS.

We used the Keras API [44] to implement our deep learning models. Keras is a open source high-level API built on top of the machine learning platform Tensorflow. We mainly use the Keras functional API to build all of our models. This API provides a convenient way of creating linear and non-linear models with different deep learning layers.

Chapter 4

Implementation & Materials

In this chapter we present our materials and implementation of the methodology in chapter 3. First we introduce the training and test datasets used in this project. Then we present the preprocessing pipeline for well log data. Next, we introduce the implementation of the four deep learning algorithms. Finally, we present how we generate ground truth labels for the test dataset, and our two approaches to model evaluation.

4.1 Preprocessing

4.1.1 Dataset

The data used for training purposes in this project is confidential and provided by Logtek AS. All data is collected from wells on the NCS. The full training dataset includes 139 well logs from a multiple wells in different oil fields. The only information we are able to share is the number of well logs and the general statistics presented in table 4.1. It is worth noting that observations with missing values are not considered during the calculation of these values.

Feature	Min	Max	Mean	Std
DEPTH	220.68	7093.00	2386.90	1086.12
AC	43.23	286.04	104.52	27.48
ACS	76.71	798.43	232.82	103.83
BS	6.00	17.5	10.52	2.45
CALI	2.00	27.93	10.69	2.65
DEN	1.15	3.65	2.37	0.21
DENC	-1.58	6.73	0.02	0.05
GR	0.0	953.73	74.99	35.22
NEU	-0.02	7.69	0.33	0.15
PEF	-59.04	137.44	4.91	2.44
RMED	0.07	100000.0	17.34	450.87
RDEP	0.07	100000.0	13.76	246.68

Table 4.1: Statistics from training dataset.

For testing and visualization purposes we utilize the publicly available Volve dataset provided by Equinor and their partners [45]. The Volve oil field is located 200km west of Stavanger and was in production for eight years between 2008 and 2016 [46]. The dataset is the most comprehensive publication of well log data from the NCS, and contains roughly 40000 files from the Volve oil field. We define the well log *15/9-F-11 T2* as our test dataset for this project. Data from this well log will be used for all numbers and figures in the report, unless otherwise specified. One of the interesting aspects of this well log is that the wellbore is drilled with an inclination, and is almost horizontal at the end of the log. The same general statistics about the test dataset is presented in table 4.2.

Feature	Min	Max	Mean	Std
DEPTH	2582.90	4513.00	3528.68	554.74
AC	48.93	136.25	74.19	13.38
ACS	74.82	388.84	138.15	37.74
BS	8.50	8.50	8.50	0.00
CALI	8.50	8.99	8.68	0.06
DEN	2.09	3.00	2.50	0.13
DENC	-0.00	0.20	0.06	0.01
GR	0.84	437.82	27.60	35.23
NEU	-0.00	0.49	0.14	0.09
PEF	4.84	11.54	7.73	1.09
RMED	0.12	46.34	3.72	3.08
RDEP	0.07	62290.77	9.42	511.89

Table 4.2: Statistics from test dataset.

4.1.2 Data Collection

This section describes how the raw composite logs are handled. Log I/O is used to read files in LIS, LAS and DLIS format. Since Log I/O is written in Java, all following steps described in this section are also done in Java as opposed to Python which is used for the remainder of the project. The following steps applies to both the training and test set.

UoM is used to perform unit conversion on the features. This ensures that the same feature from multiple well logs use the same unit of measure. Unit conversion is a necessary step as a deep learning models does not distinguish between the unit of measure. It would be problematic if one composite log in the training set used meters as the unit for DEPTH while another log used tenths of inches. The model would not understand that 2000 m and 787401.575 in/10 are the same.

As described in section 3.1.3, a log curve may have many aliases. The feature names are therefore renamed to conform to table 3.1. It would be highly inconvenient if the training set contained multiple features representing the same type of data.

The features we use are further detailed in section 4.1.3. Any additional features are dropped. The remaining features are then reordered to a desired order. This is to ensure that the n-th column in the dataset always correspond with the n-th feature.

Finally, a composite log is converted to a CSV format so it can easily be handled in Python. However, if at least one step above could not be performed, then the composite log is not included in the training set. It is important to mention that the composite logs have not been merged into a single file or dataset. This is beneficial as each file can be loaded individually as a separate time series or combined when needed.

4.1.3 Feature Selection & Transformation

All features described in section 3.1.2 (except DEPTH) are used in this project. The selection of these features are based on input from the petrophysicists at Logtek, and their frequent use in petrophysical analysis. Additionally, these features are often present in composite logs, increasing the amount of available data for the training set. Features such as Thorium and Uranium measures the presence of these elements in parts per million and could potentially be interesting to include. However, including these features would significantly reduce the training set because they either appear too infrequent in composite logs or contain too much missing data. Features such as rate of penetration are excluded because it relates to how fast the drill is drilling and not to the lithology itself. The measured depth is not included as it is not necessarily indicative of which type of rock formation will be found. Additionally, the wellbore's trajectory occasionally ends up going nearly horizontal. In those situations, the true vertical depth remains nearly constant. Thus, the measured depth is only used as an index which does not give information about the lithology at hand.

Figure 4.1 shows the distribution of RMED values in the training set. The majority of observations have a value between 0 and 10 ohmmeters, but can occasionally have a value of tens of thousands ohmmeters. RMED and RDEP are therefore typically represented on a logarithmic scale during petrophysical analysis. A logarithmic transformation is applied to these features using the base 10 logarithm. The benefit of applying logarithmic transformation is that the minimum and the maximum value of the feature

are much closer to one another, which is important when scaling the data (see section 4.1.5). Additionally, the logarithmic transformation makes the distribution appear to be normal rather than exponential-like. The logarithmic transformation change the range of values in RMED from $[0.07, 100000]$ to $(-1.155, 5]$.

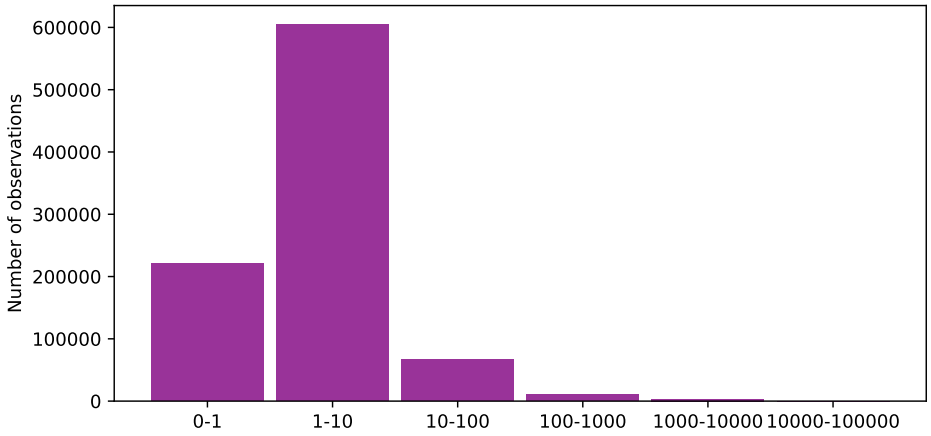


Figure 4.1: Distribution of values in RMED.

BS is used in conjunction with CALI to detect shale swellings and cave-ins along the wellbore path. By itself, BS only tells the diameter of the drill bit and does not provide information about lithology. BS should therefore not be considered by any of our models. CALI measures the diameter of the wellbore, i.e. the bit size plus the distance between the drill bit and the walls. The issue of including CALI is that its value is always influenced by BS. If BS is decreased by 9 inches, then CALI is reduced by the same amount. To prevent the models to learn this characteristic, we introduce a new feature CALI-BS which is the difference between CALI and BS. Both BS and CALI are then dropped. The models should now be able to use the information CALI provides regardless of the actual bit size. Figure 4.2 shows a visualization of the issue using fabricated data. The caliper feature is always slightly higher than the bit size. When the bit size is reduced the amplitude of the caliper feature is reduced by the same amount. The green line shows how the new feature CALI-BS is unaffected by the change in bit size.

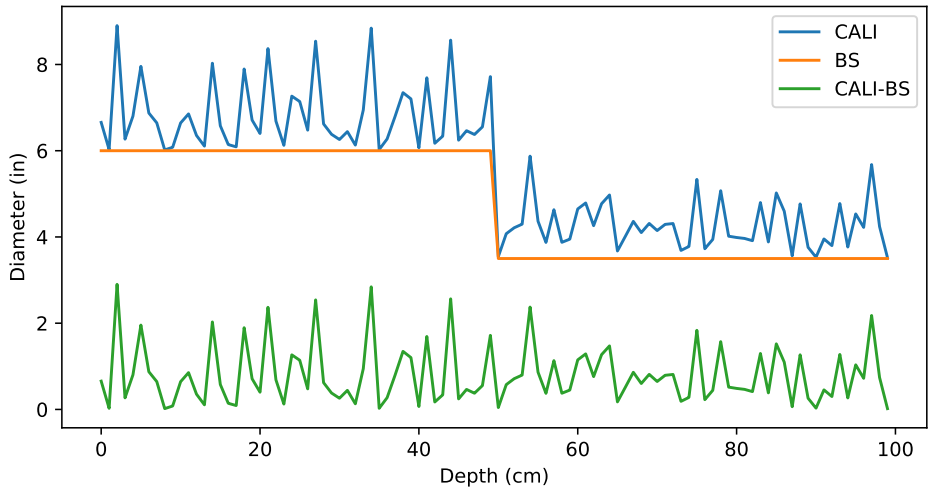


Figure 4.2: Visualization of the difference between BS, CALI and CALI-BS.

All other features in the dataset (AC, ACS, DEN, DENC, GR, NEU & PEF) are not transformed or altered in any way. Together with RMED, RDEP & CALI-BS, these features constitutes the selected feature set.

4.1.4 Missing Values

The missing values in the dataset are assumed to be NMAR, as the most likely reason why data is missing (in composite logs) is due to sensors being purposely turned off. Disabling sensors in uninteresting areas is a common method of saving costs for the service company. However, we also assumed the existence of some missing data due to wellbore problems, sensors malfunctioning or removal of invalid/erroneous data. All these scenarios could be either MAR or MCAR depending on the situation.

The first step in the the process of handling missing values was to analyze where they occur. This analysis was necessary for deciding which missing value approach was suitable for the dataset. The first analysis showed that 65% of observations contains at least one missing value, which means the common problem of large amounts of missing data in well log data is also reflected in our dataset. Figure 4.3 presents the proportion of missing values

per feature, and shows that the issues is persistent across multiple features. Only a hand-full of features have less than 5% missing values, while half the features have more than 40%.

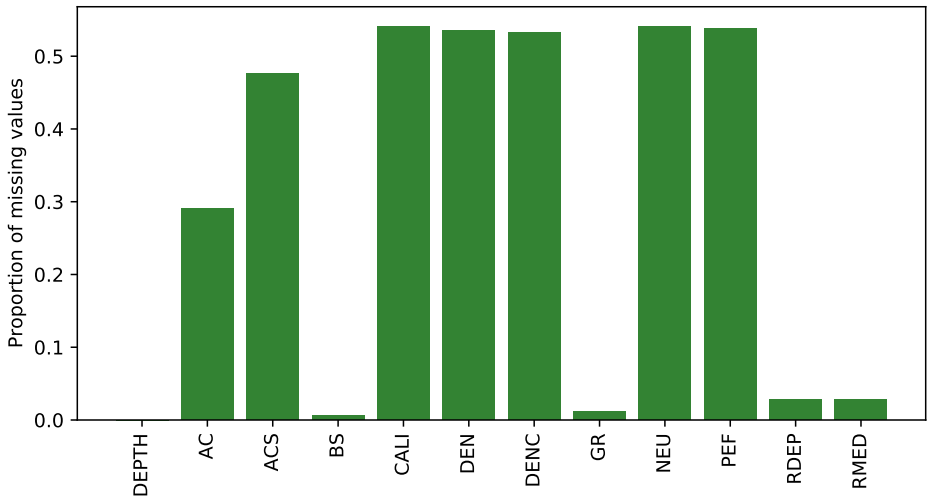


Figure 4.3: Proportion of missing values per feature.

Figure 4.4 shows that incomplete observations typically contain missing data in five to seven features. This supports the assumption that data is NMAR as the majority of incomplete observations contains missing data in multiple features.

During development, several strategies were employed with varying levels of success. Single imputation with mean or median, as well as the time series specific LOCF and NOCB methods, proved to be unsatisfactory on our dataset. The main reason for this is due to the nature of the problem we are trying to solve. All these methods impute fabricated data into the dataset and is not interpreted any different than complete observations. Considering the substantial amount of missing data, imputation heavily impacts the "ground truth" of what is considered normal behavior. This will in turn work against the purpose of the anomaly detection models.

Due to the excessive amounts of missing values, the selected approach for handling missing values is to remove the associated observations. An important consequence of this approach is that the well log is no longer a single continuous time series. When an observation or section of observations are

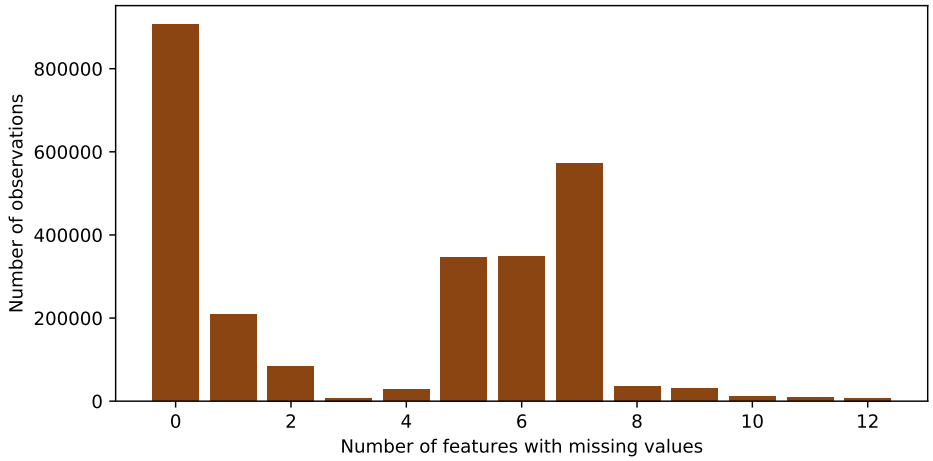


Figure 4.4: Proportion of missing values per observation.

removed in a time series, the gap effectively splits the well log into two time series.

Our solution to this problem is to split each well log file into several continuous subfiles. The result of this process is multiple disjoint datasets, where each subfile is considered and handled as a unique time series. The purpose of this step is to avoid introduction of fabricated data and retain the context between complete observations. Figure 4.5 shows a clear visualization of the process of removing missing values and the resulting split subfiles for a single well log. After removing all the missing values we are left with 1212 subfiles with an average length of 750 observations. This leaves us with over 900 000 complete observations that are used for training the machine learning models. The test dataset is split into three subfiles with a total of 18703 observations.

4.1.5 Normalization

The dataset is normalized using the min-max normalization approach described in section 3.2.2. The feature scaling is performed after the removal of missing values and feature transformation. We extract the global minimum and maximum value of each feature in order to calculate the scaled

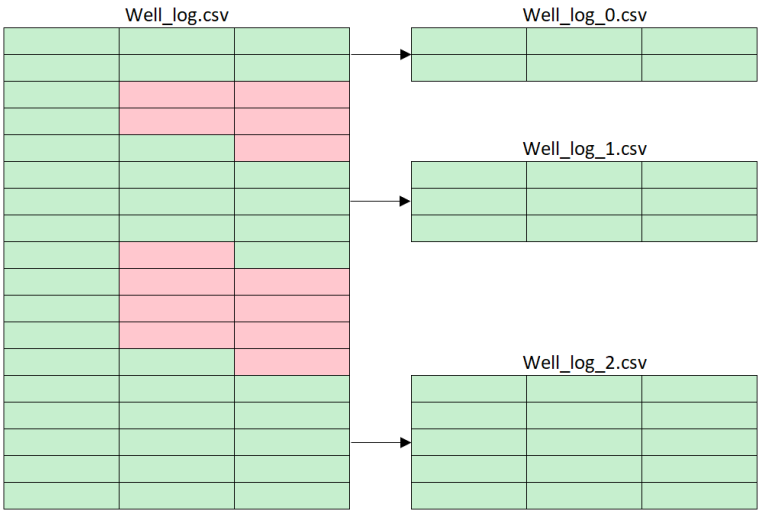


Figure 4.5: The process of removing missing values and splitting composite logs into multiple subfiles.

values. A scaler object was created and saved to transform the subfiles of both the training and test datasets when needed. The impact of removing missing data, feature transformation and feature scaling on the training and test datasets are presented in table 4.3 and 4.4.

Feature	Min	Max	Mean	Std
AC	0.00	1.00	0.25	0.11
ACS	0.00	1.00	0.22	0.14
DEN	0.00	1.00	0.49	0.08
DENC	0.00	1.00	0.19	0.01
GR	0.00	1.00	0.08	0.04
NEU	0.00	1.00	0.05	0.02
PEF	0.00	1.00	0.33	0.01
RMED	0.00	1.00	0.24	0.08
RDEP	0.00	1.00	0.25	0.08
CALI-BS	0.00	1.00	0.27	0.03

Table 4.3: Statistics from the scaled training dataset.

Feature	Min	Max	Mean	Std
AC	0.02	0.38	0.13	0.06
ACS	-0.00	0.43	0.09	0.05
DEN	0.37	0.74	0.54	0.05
DENC	0.19	0.21	0.20	0.00
GR	0.00	0.46	0.03	0.04
NEU	0.00	0.07	0.02	0.01
PEF	0.33	0.36	0.34	0.01
RMED	0.03	0.46	0.25	0.06
RDEP	0.00	0.97	0.26	0.07
CALI-BS	0.26	0.28	0.27	0.00

Table 4.4: Statistics from the scaled test dataset.

4.1.6 Reshaping

The final step is to scale and the reshape the datasets to fit the input of the models. The datasets must be shaped differently based on the different input shapes of the models. The reshaping process is therefore explained in more detail for each model in section 4.2.

4.2 Models

The models presented in this section can be separated into reconstruction and prediction based models. AE and VAE are reconstruction models as the input and output correspond to the same observation. The reconstruction is in a sense a prediction that AE or VAE makes. Thus, the prediction from AE and VAE models is analogous with reconstruction. LSTM and DeepAnT are predictive models that predicts the next observation based on the *timesteps* previous observations. What all four model types have in common is that they are self-supervised and the output's format & shape. The output of a single prediction is always an observation. This implies that DeepAnT's anomaly detector module can also be applied to the other model types. All models can therefore be evaluated using the same method.

4.2.1 Autoencoder

Model Implementation

The AE model comprise only of fully connected layers. The minimum number of layers is three, where only the input, latent space and output layers are defined. The number of neurons in the input and output layers are equal to the number of features in the dataset. The size of the latent space layer can be configured. The number of intermediate layers in the encoder part can also be configured along with their respective number of neurons. The order of these layers is reversed in the decoder. For example, if there are 10 features, the dimension of the latent space is 2 and intermediate layers given as $[6, 4]$, then the autoencoder has the following shape $[10, 6, 4, 2, 4, 6, 10]$. The latent space layer and all intermediate layers use the ReLU activation function. The output layer uses the sigmoid activation function, while no activation function is used on the input layer.

Reshape Input Data

Since the autoencoder only evaluates one observation at a time, it has no notion of time. The dataset therefore does not have to be a time series. This simplifies the learning process as the input dataset can be constructed by concatenating all preprocessed composite logs. Obtaining a validation split is the only slight implication. The validation split is obtained by taking 10% of data from each subfile to ensure that validation occurs for all wells and not just the last portion of the complete concatenated dataset. Using the last portion of the complete dataset would mean that validation only has data from the final logs in the dataset. The AE model's input shape is (n, org_dim), which the dataset fortunately has. Thus, no reshaping is required.

4.2.2 Variational Autoencoder

Model Implementation

The variational autoencoder shares the overall structure with a regular autoencoder described in 4.2.1. What separates VAE from AE is the inclusion of sampling layers and the incorporation of Kullback Leibler divergence when computing the loss. Additionally, the encoder and decoder are created separately such that it would be possible to obtain the latent representation of the input to study the latent space. Studying the latent space is out of scope for this project and is primarily interesting for future work. The decoder is concatenated to the encoder during the learning phase and are thus trained in tandem.

As described in section 3.4.2, the encoder in a variational autoencoder encodes the input as a distribution over the latent space z. The latent representation of the inputs is obtained by sampling from this distribution using its mean μ and standard deviation σ. μ and σ are implemented as two fully connected layers that are connected to the last intermediate layer. These two layers are not connected to each other and the number of neurons are equal to the number of dimensions in the latent space. The latent space layer z is connected to both μ and σ. The output of z is obtained by sampling from a normal distribution with μ as the mean and σ as the standard deviation. The complete VAE model is then constructed by connecting the decoder to z. The decoder is not different from a decoder in a regular autoencoder.

Reshape Input Data

Since the general shape of a VAE is identical to an AE, everything mentioned in section 4.2.1 also holds for VAE. Thus, the only reshaping required is to extract a validation set and concatenate all the subfiles into a single dataset.

4.2.3 LSTM

Model Implementation

The implementation of the LSTM model is based on the Keras functional API. The model is comprised by one or more LSTM layers and a Dense layer. Each LSTM layer in the model can be viewed as a separate RNN with the LSTM architecture described in section 3.4.3. The input of every LSTM layer is a tensor with the shape ($batch\_size$, $timesteps$, $original\_dim$). If there are multiple LSTM layers in the model, the additional parameter $return\_sequences$ is set to $True$. If this parameter is set to $False$, the LSTM layer will return the final hidden state of each neuron. Thus, the output of this layer has the shape ($n\_neurons, original\_dim$). However, the subsequent LSTM layer requires the input to follow the previously mentioned tensor shape. The $return\_sequences$ parameter solves this problem by returning the hidden state at every timestep for each neuron, effectively returning the desired input shape. It is important to note that the hidden state of a single neuron is an array in this implementation. The $return\_sequences$ parameter is not required to be set for the final LSTM layer as we're only interested in the final hidden state.

Each LSTM layer is assigned to use the same recurrent dropout probability. For the majority of deep learning models, a separate dropout layer is used to reduce overfitting by randomly dropping output from random neurons. However, the recurrent dropout parameter defines the probability of dropping the internal connection in the LSTM cell's input gate when computing the cell state. This method has shown to outperform the traditional dropout layer approach in LSTM networks [47, 48]. The final layer in the model is the Dense layer, which is always defined to be the size of the output dimension. This layer is used to reduce the number of features down to the size of the output. The output of this layer is used as the prediction for the following observation. The loss function MAE and Adam optimization with default learning rate (0.001) is used to compile the model.

Reshape Input Data

Since our LSTM model is a self-supervised deep learning algorithm, the labels used during training needs to be constructed from the original dataset D. The dataset D contains n records r of length $original\_dim$ and is denoted as in (4.1a). LSTM require $timesteps$ subsequent records to predict the next timestep. For example, with $timesteps = 3$, then $\{r_1, r_2, r_3\}$ is used to predict r_4 and $\{r_2, r_3, r_4\}$ is used to predict r_5. Using the notation in (4.1d) and (4.1f), X_i is used to predict y_i respectively. The input for the LSTM model is denoted by the set X (4.1c) which contains n_{out} matrices with the shape $(timesteps, original\_dim)$. Accordingly, Y (4.1e) is a set of labels which are used to compute the loss of a training pass. Both X and Y contain n_{out} (4.1b) elements. Consequently, predictions cannot be made for r_i where $i \leq timesteps$.

$$D = \{r_1, r_2, \ldots, r_n\} \tag{4.1a}$$
$$n_{out} = n - timesteps \tag{4.1b}$$
$$X = \{X_1, X_2, \ldots, X_i, \ldots, X_{n_{out}}\} \tag{4.1c}$$
$$X_i = \{r_i, r_{i+1}, \ldots, r_{i+timesteps-1}\} \tag{4.1d}$$
$$Y = \{y_1, y_2, \ldots, y_i, \ldots, y_{n_{out}}\} \tag{4.1e}$$
$$y_i = r_{i+timesteps} \tag{4.1f}$$

As described in section 4.1.4, each composite log may be split into multiple datasets. A dataset by itself is equivalent with the notation of D in (4.1a). Each dataset in the collection is considered disjoint time series and needs to be handled accordingly during training. The Keras library allows for a model to be fit consecutive times without resetting the weights. Thus, the model is fit once for each X and Y pair derived from the collection of datasets. This constitutes one epoch. Code segment 4.1 shows a simplified training loop where the variable *datasets* is a collection of (X, Y) tuples. The hidden state of the LSTM cell is reset after fitting each dataset. The purpose of this step is to avoid using the hidden state from the previous dataset as an input to the next. This ensures the datasets are considered as separate disjoint time series in the LSTM network.

```
for epoch in range(n_epochs):
    for X, Y in datasets:
        model.fit(x=X,y=Y, batch_size=len(X))
        model.reset_states() # for LSTM only
```

Code segment 4.1: "Python code showing the training process with disjoint datasets."

4.2.4 DeepAnT

Model Implementation

Our implementation of DeepAnT mimics the general architecture detailed in section 3.4.4. The DeepAnT model comprise of Keras layers in the following order: Conv1D, MaxPooling1D, Conv1D, MaxPooling1D, Flatten, Dense, Dropout and Dense. All convolutional and dense layers use the ReLU activation function. The input shape of the first convolutional layer is a tensor with the shape $(batch\_size, timesteps, original\_dim)$. Both convolutional layers have a kernel size of k and C_{out} output filters. Both max pooling layers has a pool size of two. After the second pooling layer, the shape is flattened to fit in a dense layer. The first dense layer has a size of $dense\_dim$. The dropout layer has a dropout rate of 25%. The final dense layer has $original\_dim$ neurons as we are trying to predict the observed data in the next timestep.

Reshape Input Data

Like LSTM, DeepAnT is also a self-supervised deep learning algorithm and has the same input shape. Thus, reshaping the input of DeepAnT is identical to the reshaping for LSTM described in section 4.2.3. The only difference between LSTM and DeepAnT is the training loop. DeepAnT has no internal hidden state that need to be reset after fitting the model with a dataset.

4.3 Model Evaluation

In this section we present how the results are produced and used. Furthermore, we describe how the output of our results would be used in a real scenario. Then we describe how we generate labels such that we can compute evaluation metrics used to evaluate the performance of the models in the Results chapter.

4.3.1 Obtaining Points of Interest

The test set is processed using the same procedure and feature scaler as the training set described in section 4.1. Thereafter, the test set is reshaped to fit the selected model. For the DeepAnT and LSTM models the test set is reshaped to a set of (X, Y) pairs as described in 4.2.3. Since AE and VAE is a one to one reconstruction algorithm there is no need to reshape the test set. Additionally, we have $Y = X$ for autoencoders. It is worth noting that Y is a matrix for any model where the number of columns is equal to the number of input features. The number of rows may however change due to the *timesteps* parameter for DeepAnT and LSTM. The next step is to feed the test set to the models and obtain the output Y', then we compare it with Y using the Euclidean distance. In section 3.4.4 the Euclidean distance is defined between two scalars and not between two vectors. That definition is thus not applicable to our multivariate dataset. How we calculate the Euclidean distance for a multivariate use case has therefore been defined in (4.2). Here y_i is an actual observation and y'_i is the corresponding prediction.

$$ED(y_i, y'_i) = \sqrt{\sum_{j=1}^{n}(y_{i,j} - y'_{i,j})^2}, \quad y_i \in Y,\ y'_i \in Y' \tag{4.2}$$

Figure 4.6 shows the Euclidean distance for each observation in the test dataset. Appendix D include ED plots for the best performing configuration of each model. The different colors indicate the sections of the test set where all features were present. This representation visually shows where a model can make predictions. Using a different subset of features may expand or

even merge these sections.

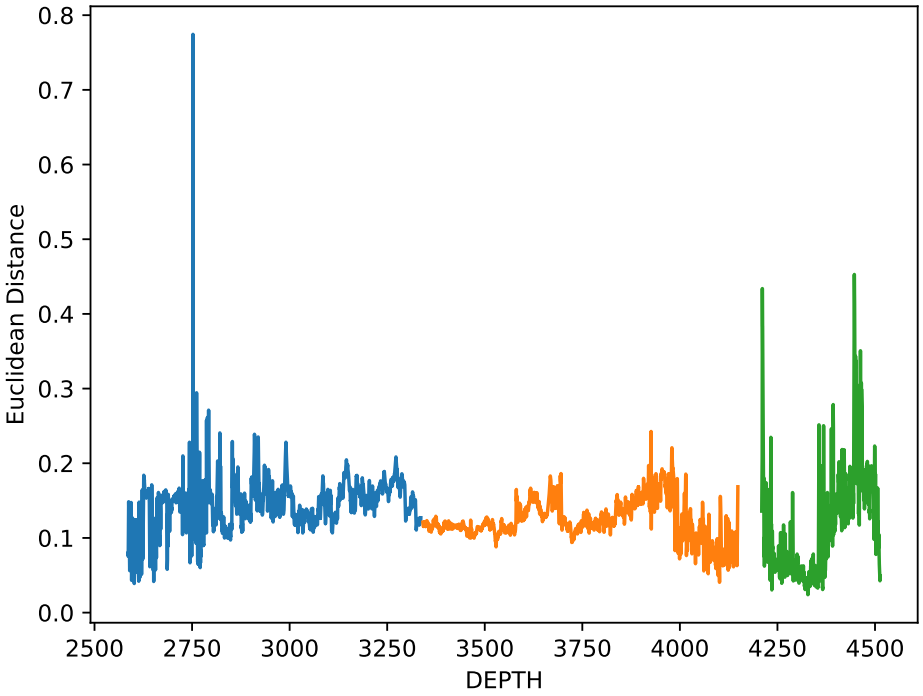


Figure 4.6: Example of Euclidean distance between original and predicted data for all observations in the test dataset.

As described in section 3.4.4, DeepAnT marks observations as anomalous if the ED is greater than a certain threshold. We employ the same principle for determining anomalous observations for all models because their output format is all the same. However, we look at the change in ED instead. Throughout experimentation, we consistently observed that certain areas have a higher Euclidean distance on average compared to other areas. Upon deeper inspection it seems that these areas correspond to lithologies that the models have difficulties in predicting. Thus, when ED is large but seemingly constant, it appears that such an area is not anomalous. If the threshold was set directly on the Euclidean distance as proposed by Munir et al. [39], there is a chance that the whole of such an area will be marked as anomalous. By looking at the change in ED instead, the edges of these areas (or plateaus) will have a higher value than in the center. Thus, the

change in ED appears to reveal transitions between lithologies. Figure 4.7 shows the change in ED for each observation in the test dataset. Similar plots for the best performing configuration of each model are included in appendix E.

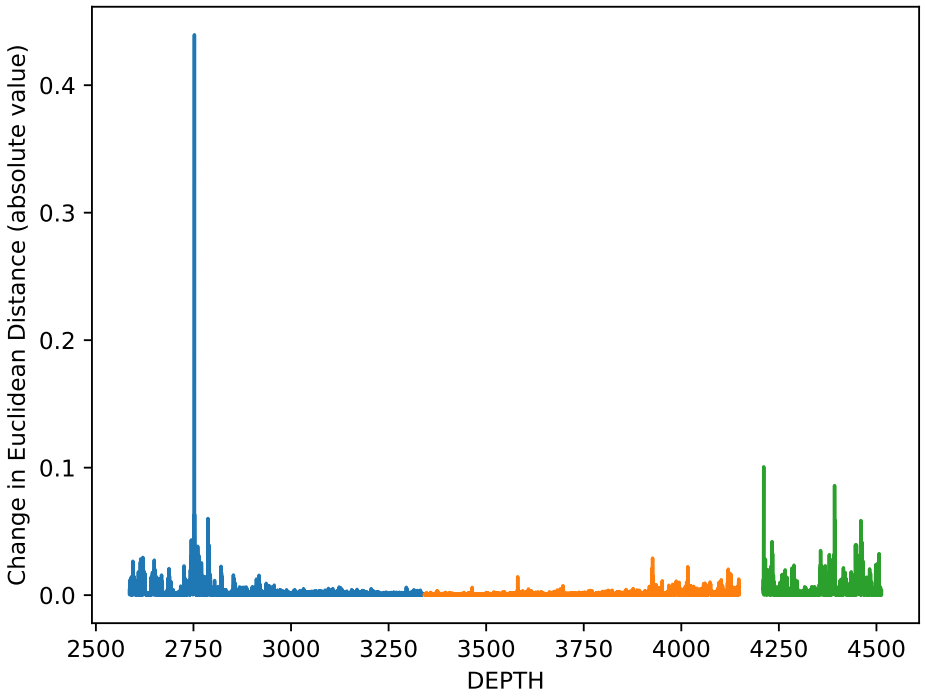


Figure 4.7: Example of change in Euclidean distance between original and predicted data for all observations in the test dataset.

The threshold is chosen to be a percentile of the change in Euclidean distance. Somewhere between the 80^{th} and the 99^{th} percentile should be a suitable threshold. An observation with a change in ED larger than the threshold is marked as a Point(s) of Interest (POI). Lowering the threshold below the 80^{th} percentile will increase the number of true & false positives positive predictions. Conversely, increasing the threshold yields a higher number of true & false negative predictions. Thus, adjusting the threshold directly influence the number of type I & II errors by increasing one of them and decreasing the other. The importance of precision versus recall is therefore imperative in deciding a suitable threshold.

In addition to POIs, it is useful to map the change in Euclidean distance to their corresponding percentile as a new output feature. The percentile then conveys the importance of a observation in terms of a score. Looking at this score in conjunction with points of interest may provide additional insight to a petrophysicist compared to looking at the points of interest alone.

Figure 4.8 shows how we envision the output (3 rightmost columns) of the models can be used to enhance the normal workflow of a petrophysicist. The POI uses the 96^{th} percentile as the threshold. Additionally, the corresponding percentile is shown for all observations higher than the 80^{th} percentile. With this column a petrophysicist can see which non-POI observations are noteworthy. The final column shows the Euclidean distance in red and its change in green. This column may be omitted as interpreting it requires knowledge of how the Euclidean distance is computed. It is unreasonable to assume that a petrophysicist has prior knowledge about the underlying deep learning algorithm.

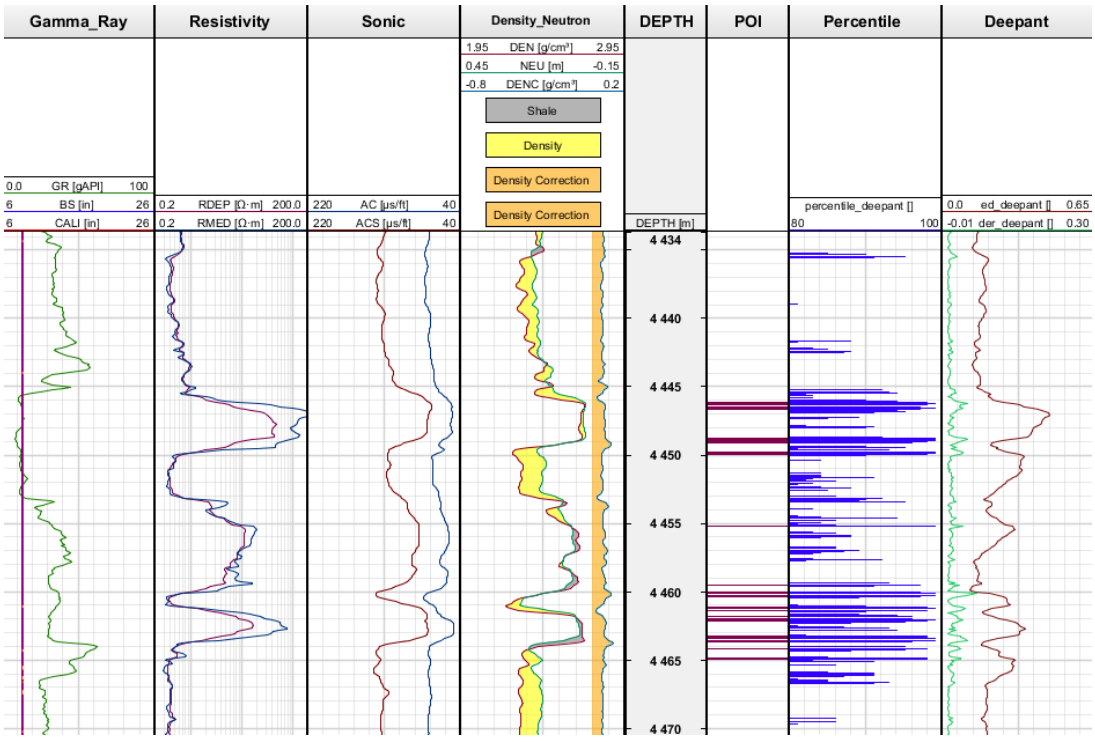
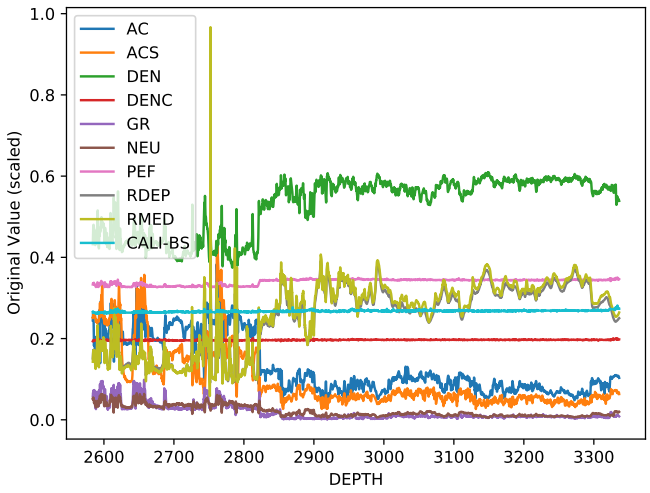
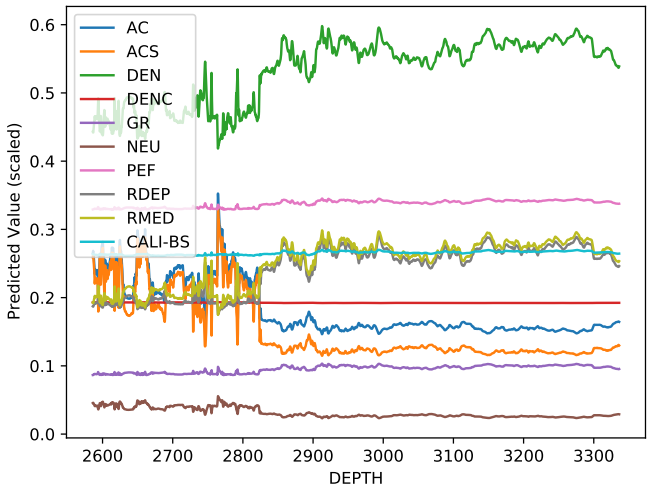


Figure 4.8: Visualization of how we envision the output of our models may be used in a petrophysical analysis. This image is from the application Log Studio.

To evaluate if a model works, we compare a plot of the scaled actual data Y and a plot of the predicted data Y'. In figure 4.9 the predictions are similar enough to the original data. This indicates that the model makes an actual prediction, and thus appears to be working.



(a) Original scaled data.



(b) Predicted scaled data.

Figure 4.9: Comparison between the scaled original and predicted data for the first subfile in the test dataset.

4.3.2 Obtaining Final Results

Euclidean distance is a metric that indicates the discrepancies between the observed and predicted data. Thus, the Euclidean distance can be used in evaluating a model. However, this method only evaluates the quality of predictions and cannot be applied in evaluating points of interest. Unfortunately, there exists no publicly available ground truth of points of interest for the well *15/9-F-11 T2*. For this reason, we sought help from petrophysicists at Logtek in creating a ground truth. They were tasked with marking single points or areas that might be interesting. The different types of areas marked by the petrophysicists are presented in the list below. Keep in mind that the term "point of interest" is vague and ambiguous, and other petrophysicists might not agree with describing the areas found as *interesting*. Additionally, we make no guarantee that the constructed ground truth is complete. It is not unlikely that some interesting areas were overlooked or wrongly included. The ground truth is included in appendix A and lists the measured depth and reason for every point of interest.

- Slate to sand
- Sand to slate
- Dense strings
- Coal
- Slate to calcite
- Calcite to slate
- Hydrocarbons
- Mudcake
- Washout
- Well deviation

The predictions are transformed into a vector of ones and zeros which corresponds to positive and negative predictions respectively. Points of interest are positive predictions. The ground truth is transformed into a vector of the same format and length except ones and zeros corresponds to actual positive and negative instances respectively. The predictions and the ground truth are then compared element-wise to construct a confusion matrix. This is only done for sections in the test set where all features are present to show the performance a model's predictions. The prevalence of the test dataset is very low, implying that the ground truth contains a minuscule amount of actual points of interest. This method is exceedingly strict as the depth of a predicted point of interest must equal the depth of an actual point of interest in order to become a true positive. Predicting a point of interest a mere 10 cm before or after the actual point of interest yields a false positive. Additionally, there is room for human error in the ground truth, meaning that a point of interest may be defined at a slightly wrong depth. For these reasons, we devise two methods for comparing predictions and the ground truth.

The first method is more lenient and expands the points in the ground truth by setting all records within a window to true instances. For example, if the record at depth 2400.3 was a positive instance, then all records within the range $[2399.8, 2400.8)$ (excluding 2400.8) will also be considered positive instances with a window of length 1. The predictions remain unaltered. Computing the confusion matrix is identical to the strict method.

The second method downsamples both the predictions and the ground truth to reduce the number of instances. This is equivalent to max pooling with a window size and stride length of one meter (10 records). Before the downsampling, the predictions and the ground truth are zero-padded at both ends. The first value corresponds to a depth of $xx.0$ and the last value corresponds to a depth of $xx.9$. Thus, values within the range $[xx.0, xx.9]$ are reduced to a single element with a value equal to the maximum within that range. The notation xx refers to the integer value of a depth. For instance, if a prediction at depth 2400.3 is positive, then the entire range $[2400.0, 2400.9]$ is reduced to a single point of interest at depth 2400.0. As a result, the number of entries in the confusion matrix decreased significantly.

DEPTH	Prediction	Ground Truth	Prediction	Lenient Ground Truth	Downsampled Prediction	Downlampled Ground Truth
2399.8	True	False	True	False	True	False
2400.1	True	False	True	False		
2400.4	True	False	True	True		
2400.7	True	True	True	True	True	True
2401.0	False	False	False	True		
2401.3	False	False	False	False		
2401.6	False	False	False	False		
2401.9	False	False	False	False	False	False
2402.2	False	False	False	False		
2402.5	False	False	False	False		
2402.8	False	False	False	False	False	False
2403.1	False	False	False	False		
2403.4	False	False	False	True		
2403.7	True	True	True	True	True	True
2404.0	True	False	True	True		
2404.3	True	False	True	False		
2404.6	True	False	True	True		
2404.9	False	True	False	True	True	True
2405.2	False	False	False	True		
2405.5	False	False	False	False	False	False

	Strict Confusion Matrix		Lenient Confusion Matrix		Downsampled Confusion Matrix	
	PP	PN	PP	PN	PP	PN
P	2	1	5	4	3	0
N	6	11	3	8	1	3
Accuracy	0.65		0.65		0.86	
Precision	0.25		0.625		0.75	
Recall	0.67		0.56		1.0	
F1	0.36		0.59		0.86	
MCC	0.23		0.29		0.75	
Prevalence	0.15		0.45		0.43	

Figure 4.10: Demonstration of the lenient and downsampling methods.

Figure 4.10 illustrates the strict, lenient and downsampling methods for comparing predictions and the ground truth. Additionally, the confusion matrix and performance metrics are shown for each method to demonstrate their impact. The predictions and ground truth presented in the figure are fictitious and was chosen only for illustration purposes.

The lenient method has a significant uplift in prevalence as 6 negative instances in the ground truth were converted to positive cases. The figure also shows that the predictions remain unaltered for the lenient method.

The reason for not altering the predictions is that by expanding the points of interest in the ground truth, some false positives are converted to true positives. If the predictions were also expanded upon, then some true negatives would be converted to false positives. The downside of this method is that it may introduce more false negatives. However, the expanded points in the ground truth increases the chances that at least one actual positive instance in the area corresponds to a true positive. Thus, the probability for alerting a petrophysicist of a potential interesting area has increased. The additional type II errors surrounding the area are of little importance as the petrophysicist only needs to be alerted of the area. It is important to emphasize the lenient method is only lenient for the evaluation. The predictions themselves are not more lenient compared to the strict method.

The increase in prevalence for the downsampling method is due to a single actual positive instance converting multiple actual negative instances. A downsampled instance in the ground truth only remain negative if all instances in that downsampling window were also negative. The number of true negatives is therefore greatly reduced. One advantage the downsampling method has over the lenient method, is that a predicted positive area does not need to overlap with an actual positive instance. This is made apparent in the interval $[2404.0, 2404.9]$ where the prediction at depth 2404.9 is negative yet the downsampled prediction is positive for this area, resulting in a true positive for this area. The lenient method instead has two true positives, one false positive and one false negative for this area. Nevertheless, both the lenient and downsampling methods are superior compared to the strict model, which is therefore excluded in any models' result section. The lenient method is set as the default method and is used unless otherwise specified.

Chapter 5

Results, Discussion & Analysis

In this chapter we present the result from the two evaluation approaches for all our models. We run twelve different configurations of each models and discuss their performance. Additionally, we discuss and compare the difference between the approaches, and rank the overall performance of our models. Finally, we identify and discuss strengths and weaknesses with our approach.

5.1 Autoencoder

5.1.1 Results

All AE evaluation configurations are presented in figure 5.1. The tuned hyperparameters for the AE models are the dimensions of the latent space, the dimensions of the intermediate layer, and the number of intermediate layers. The latent space dimensions selected in this analysis was set to $[2, 3, 4]$ to cover the most important dimensions. A dimension of 2 or 3 allows the use of 2D or 3D visualization tools to present the outcome of the latent space. The ability to visualize and analyze the latent space is a major advantage of autoencoder based models. We also decided to include a dimension size of 4 to explore if it would provide better results despite

the inability to visualize the latent space. The number of neurons in the intermediate layer was set to a variety of values between the dimensions of the input and the latent space. Additionally, we decided to include configurations with multiple intermediate layers. The number of neurons in each intermediate layer varies between configurations, but they will always be defined in descending order.

During initial testing of the model, a set of hyperparameters were found to produce the best result. From the results of this test, we decided to define these hyperparameters to be constant for each of the configurations in figure 5.1. All of the configurations use MAE as the loss function and Adam optimization with default learning rate (0.001). The tests also suggested the models should be trained for 20 epochs, as any additional training provided no additional improvements to the performance. The number of observations per batch is set to 64.

Model	**Latent Dim**	**Intermediate Dim**
A_1	2	[4]
A_2	2	[6]
A_3	2	[6, 4]
A_4	2	[6, 4, 4]
A_5	3	[5]
A_6	3	[7]
A_7	3	[7, 5]
A_8	3	[7, 5, 5]
A_9	4	[6]
A_{10}	4	[8]
A_{11}	4	[8, 6]
A_{12}	4	[8, 6, 6]

Table 5.1: Autoencoder configurations.

The result of the lenient and downsampling approach for each configuration is presented in figure 5.2 and 5.3. The table presents the best performing threshold for each configuration with multiple metrics, as well as the average performance. The definition of the best performing threshold is based on the MCC metric with five decimals.

Model	t	ACC	precision	recall	F1	MCC	prevalence
A_1	91.00	0.90	0.27	0.43	0.33	0.29	0.06
A_2	84.00	0.85	0.20	0.58	0.30	0.28	0.06
A_3	90.00	0.89	0.24	0.43	0.31	0.27	0.06
A_4	90.00	0.89	0.24	0.43	0.31	0.27	0.06
A_5	90.00	0.89	0.23	0.41	0.30	0.26	0.06
A_6	89.00	0.88	0.21	0.42	0.28	0.24	0.06
A_7	89.00	0.89	0.24	0.47	0.32	0.28	0.06
A_8	91.00	0.90	0.27	0.43	0.33	0.29	0.06
A_9	87.00	0.87	0.22	0.51	0.31	0.28	0.06
A_{10}	89.00	0.89	0.24	0.47	0.32	0.28	0.06
A_{11}	88.00	0.88	0.24	0.50	0.32	0.29	0.06
A_{12}	89.00	0.88	0.22	0.43	0.29	0.25	0.06
A_{avg}	88.92	0.88	0.24	0.46	0.31	0.27	0.06

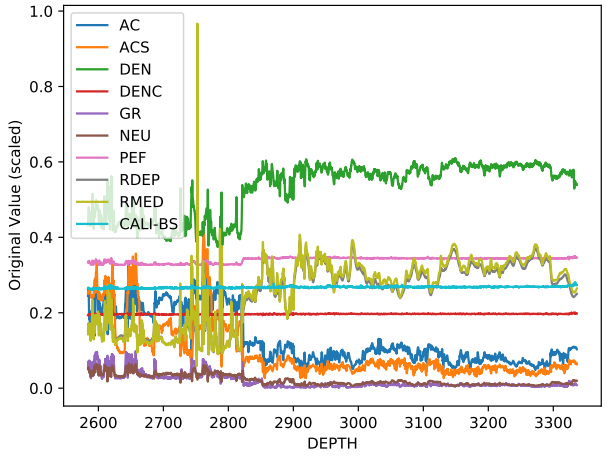
Table 5.2: Autoencoder results with lenient approach.

Model	t	ACC	precision	recall	F1	MCC	prevalence
A_1	94.00	0.83	0.30	0.65	0.41	0.37	0.09
A_2	93.00	0.82	0.30	0.73	0.43	0.39	0.09
A_3	93.00	0.83	0.30	0.71	0.43	0.38	0.09
A_4	93.00	0.82	0.30	0.71	0.42	0.38	0.09
A_5	94.00	0.84	0.32	0.68	0.44	0.39	0.09
A_6	94.00	0.84	0.32	0.68	0.44	0.39	0.09
A_7	94.00	0.84	0.32	0.70	0.44	0.40	0.09
A_8	96.00	0.89	0.44	0.62	0.51	0.46	0.09
A_9	94.00	0.84	0.32	0.69	0.43	0.39	0.09
A_{10}	97.00	0.89	0.40	0.46	0.43	0.37	0.09
A_{11}	96.00	0.87	0.36	0.55	0.43	0.38	0.09
A_{12}	96.00	0.87	0.36	0.55	0.44	0.38	0.09
A_{avg}	94.50	0.85	0.34	0.64	0.44	0.39	0.09

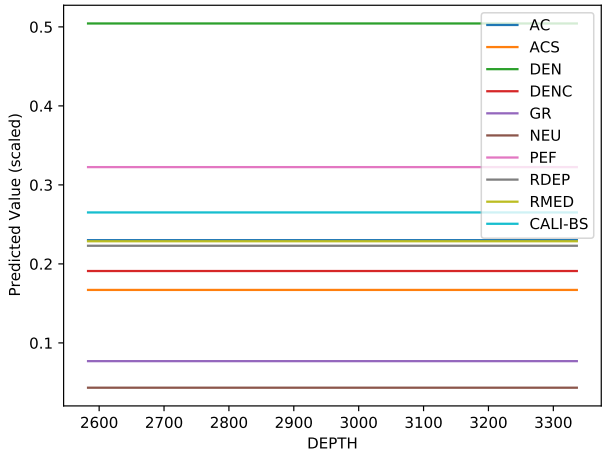
Table 5.3: Autoencoder results with downsampled approach.

5.1.2 Discussion

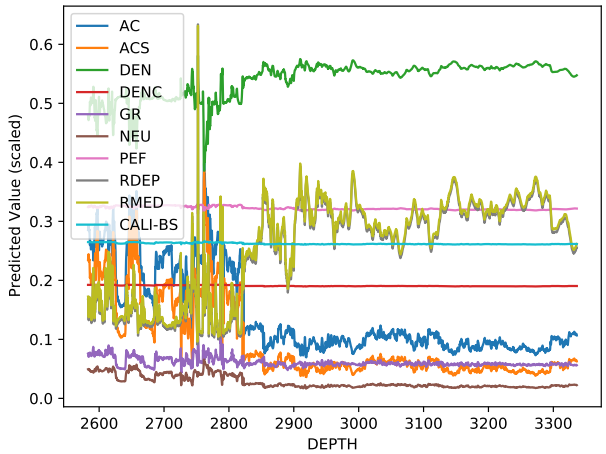
The results of the AE model shows that every configuration except one has an MCC value of 0.37-0.40 (downsampled). One of the configuration, A_8, appears to have a significantly better performance than the rest of the configurations. The output of each configuration was analyzed by comparing the original and predicted values. This analysis showed that the outlier A_8 have not produced any valid predictions. Figure 5.1 presents a comparison of the scaled original data, and predicted data from the two models A_7 and A_8. This figure clearly shows that A_8 does not perform any valid predictions. The model predicts a constant value for each feature throughout the entire test file. It appears the values are roughly equal to the mean value of each feature in the training dataset (table 4.3). This suggests the model is unable to learn from the dataset and predicts the average value to reduce the outcome of the loss function. The model A_8 is considered invalid as the model is unable to produce any valid predictions. Discussion regarding invalid models and their performance is further discussed in section 5.5.2.



(a) Original data



(b) Predicted data (A_8)



(c) Predicted data (A_7)

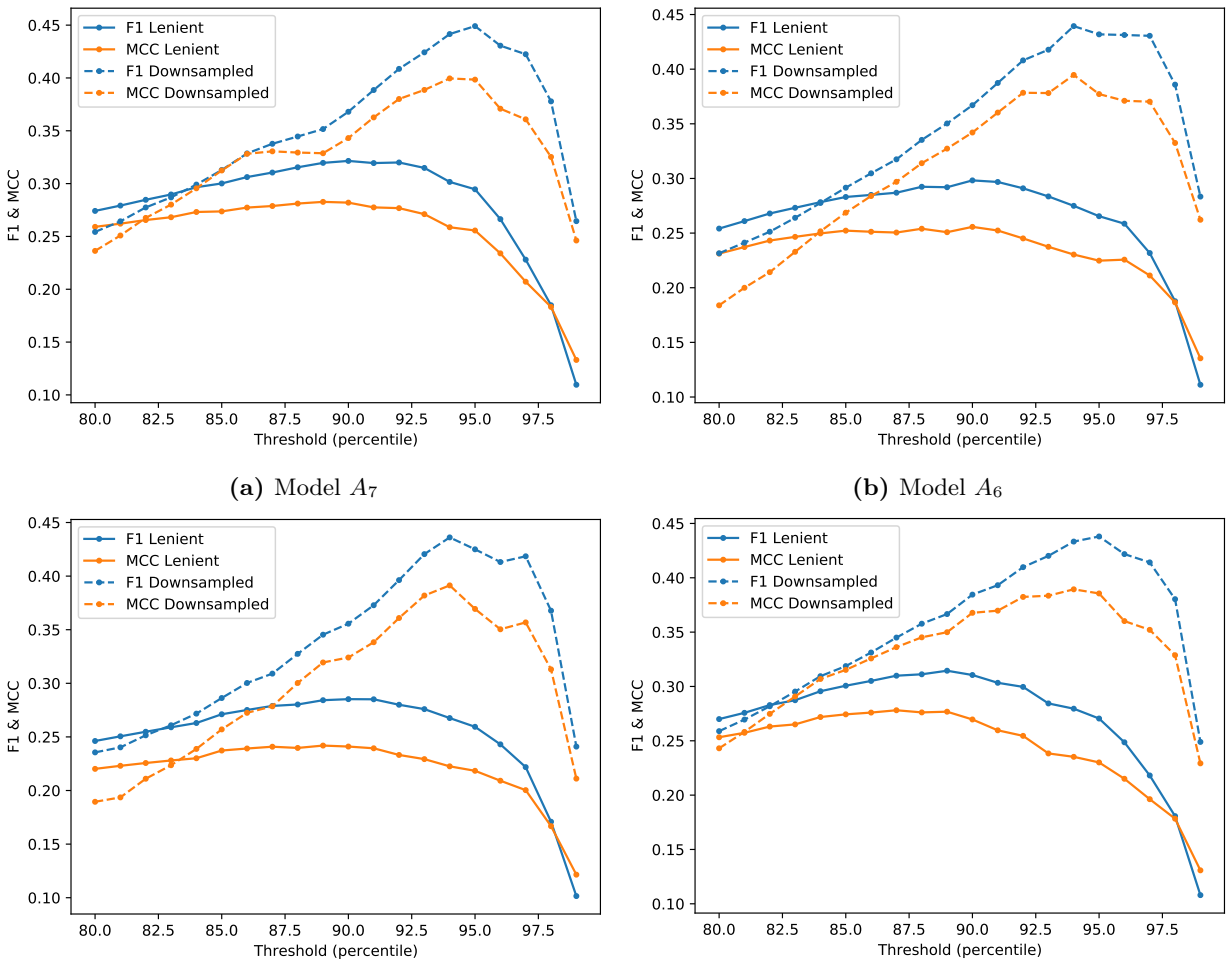
Figure 5.1: Comparison between scaled original and predicted data from A_7 and A_8.

The downsampling method appears to perform noticeably better than the lenient method for all model configurations. The results from the four best model are presented in descending order in table 5.4. All four models have very similar performance in all metrics. It is evident from the results that the autoencoder models performs best with a threshold at 94. The top three models are all configured with a late space dimension of 3, while the fourth best model has a dimension of 4. One could argue that the optimal latent space dimension is 3, but the results in figure 5.3 suggests there are only minor differences between the configurations. The number and size of intermediate layers does not appear to play a large role in the performance of the model. One could also argue the main difference in performance could be from the random weights assigned to the neuron connections.

Model	t	ACC	precision	recall	F1	MCC	prevalence
A_7	94.00	0.84	0.32	0.70	0.44	0.40	0.09
A_5	94.00	0.84	0.32	0.68	0.44	0.39	0.09
A_6	94.00	0.84	0.32	0.68	0.44	0.39	0.09
A_9	94.00	0.84	0.32	0.69	0.43	0.39	0.09

Table 5.4: The four best performing AE models (downsampled).

The MCC and F1-score of all thresholds between 80 and 99 for the four best AE models are presented in figure 5.2. The lenient approach appears to outperform the downsampling approach up until a threshold of 82-84. From this point on the downsampling approach will massively outperform the lenient approach. The lenient approach reaches its peak around 89 and gradually decreases until 97-98, where it drops dramatically. The downsampling approach reaches its peak at 94 and starts dropping rapidly with a threshold above 96.



(a) Model A_7

(b) Model A_6

(c) Model A_5

(d) Model A_9

Figure 5.2: F1 and MCC for the best performing AE models.

5.2 Variational Autoencoder

5.2.1 Results

The model configurations for the VAE model is presented in table 5.5. All the configurations are equal to the AE configurations presented in table 5.1. The number of dimensions of the latent space varies from 2 to 4, and the number of intermediate layers varies from 1 to 3. The number of neurons in the intermediate layers will be defined to be between the input and latent space dimensions, and is always in descending order in the encoder. The other constant hyperparameters are MAE as the loss function and Adam optimization with default learning rate (0.001). Like the AE models, all configurations are trained for 20 epochs with 64 observations per batch.

Model	**Latent Dim**	**Intermediate Dim**
V_1	2	[4]
V_2	2	[6]
V_3	2	[6, 4]
V_4	2	[6, 4, 4]
V_5	3	[5]
V_6	3	[7]
V_7	3	[7, 5]
V_8	3	[7, 5, 5]
V_9	4	[6]
V_{10}	4	[8]
V_{11}	4	[8, 6]
V_{12}	4	[8, 6, 6]

Table 5.5: Variational autoencoder configurations.

The VAE results for the lenient and downsampling approach is presented in table 5.6 and 5.7 respectively. Each entry in the table correspond to the configurations in table 5.5 at their best performing threshold. Threshold are selected primarily based on the value of the MCC metric with five decimals. The F1-score and precision values will be used to break ties if there are two thresholds with identical MCC value. An additional entry is added for the average performance of the VAE configurations.

Model	t	ACC	precision	recall	F1	MCC	prevalence
V_1	93.00	0.92	0.31	0.38	0.34	0.30	0.06
V_2	92.00	0.91	0.26	0.37	0.31	0.26	0.06
V_3	94.00	0.92	0.33	0.35	0.34	0.30	0.06
V_4	94.00	0.92	0.33	0.35	0.34	0.30	0.06
V_5	93.00	0.91	0.29	0.36	0.32	0.27	0.06
V_6	93.00	0.91	0.27	0.33	0.30	0.25	0.06
V_7	94.00	0.92	0.33	0.35	0.34	0.30	0.06
V_8	94.00	0.92	0.33	0.35	0.34	0.30	0.06
V_9	94.00	0.92	0.31	0.33	0.32	0.28	0.06
V_{10}	92.00	0.91	0.29	0.41	0.34	0.30	0.06
V_{11}	94.00	0.92	0.33	0.35	0.34	0.30	0.06
V_{12}	94.00	0.92	0.33	0.35	0.34	0.30	0.06
V_{avg}	93.42	0.92	0.31	0.36	0.33	0.29	0.06

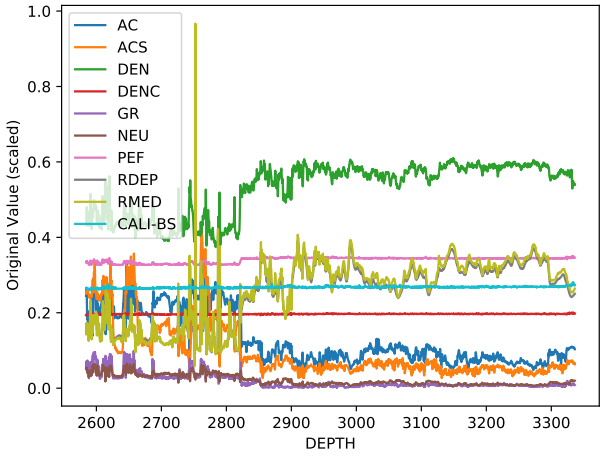
Table 5.6: Variational autoencoder results with lenient approach.

Model	t	ACC	precision	recall	F1	MCC	prevalence
V_1	95.00	0.87	0.39	0.68	0.50	0.45	0.09
V_2	98.00	0.89	0.42	0.45	0.44	0.38	0.09
V_3	95.00	0.88	0.41	0.68	0.52	0.47	0.09
V_4	95.00	0.88	0.41	0.68	0.52	0.47	0.09
V_5	96.00	0.86	0.34	0.62	0.44	0.38	0.09
V_6	97.00	0.86	0.34	0.56	0.43	0.37	0.09
V_7	95.00	0.88	0.41	0.68	0.51	0.47	0.09
V_8	95.00	0.88	0.41	0.68	0.52	0.47	0.09
V_9	96.00	0.86	0.36	0.65	0.47	0.42	0.09
V_{10}	96.00	0.89	0.43	0.64	0.51	0.47	0.09
V_{11}	95.00	0.88	0.41	0.68	0.52	0.47	0.09
V_{12}	95.00	0.88	0.42	0.68	0.52	0.47	0.09
V_{avg}	95.67	0.88	0.40	0.64	0.49	0.44	0.09

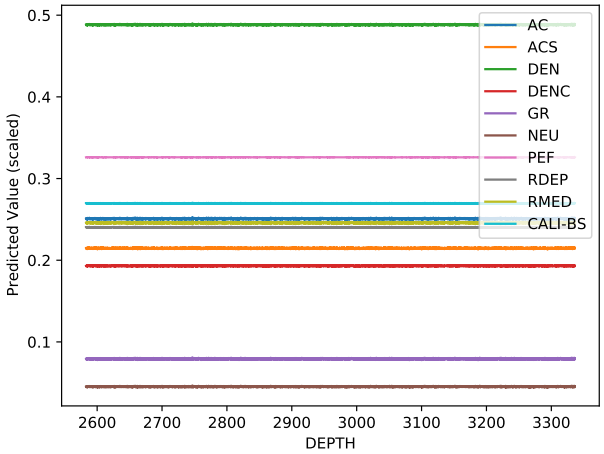
Table 5.7: Variational autoencoder results with downsampled approach.

5.2.2 Discussion

The seemingly promising performance shown in table 5.7 is misleading as all of the VAE models produce predictions of poor quality. The predictions produced by V_1 is compared against the original scaled data in figure 5.3. The predictions are constant and close to the mean of the corresponding features (table 4.3). This is the case for most VAE configurations. Some configurations have slight variations in their predictions, but are still considered of poor quality. The cause of predicting constant values is likely due to the model being unable to reconstruct drilling data and resorts to predicting constant values that minimize the loss. For this reason, none of the VAE models are valid. Discussion regarding invalid models and their performance is further discussed in section 5.5.2.



(a) Original data



(b) Predicted data (V_1)

Figure 5.3: Comparison between scaled original and predicted data from V_1.

5.3 LSTM

5.3.1 Results

All LSTM evaluation configurations are presented in figure 5.8. The hyperparameters we want to tune for the LSTM models are the number of timesteps, hidden neurons and depth of the model. The timesteps of choice are $[12, 32, 60]$ which corresponds to 1.2m, 3.2m and 6.0m distance in real life. These values for the timesteps parameter were chosen to explore the effect of having short, medium and long scope on the data. The number of hidden neurons varies based on the number of timesteps. Initial testing suggested that a hidden neuron size smaller than the number of timesteps provided the best results. The final hyperparameter we were interested in exploring was the depth of the model. Stacking two or more LSTM layers on top of each other increase the complexity of the model, but has the possibility of improving the results. The main strategy for hidden neuron size in the extra LSTM layers was to chain the previous hidden layer sizes in a descending order.

Some initial testing was performed to define a set of constant hyperparameters for each model. These tests found that the models performed best with the use Adam optimization with default learning rate (0.001) and MAE for the loss function. The initial tests also showed a tendency for the models to slightly overfit the training dataset. A decision was made to include a recurrent dropout in the LSTM layer to decrease the overfitting. The results of exploring different dropout probabilities showed a slight increase in performance with dropout set to 20%. Each model configuration is trained on four epochs as the initial tests showed little to no increase in performance past this point. Additionally, the batch size during training is set to the length of each individual subfile.

Model	**Timesteps**	**Neurons**
L_1	12	[2]
L_2	12	[4]
L_3	12	[4, 2]
L_4	12	[8, 4, 2]
L_5	32	[6]
L_6	32	[12]
L_7	32	[12, 6]
L_8	32	[24, 12, 6]
L_9	60	[12]
L_{10}	60	[24]
L_{11}	60	[24, 12]
L_{12}	60	[48, 24, 12]

Table 5.8: LSTM configurations.

The result of each configuration for both evaluation approaches are presented in table 5.9 and 5.10. The table shows the performance of each configuration on different metrics at its best performing evaluation threshold. The table also shows the average performance of the configurations for both evaluation approaches. The best performing threshold is selected based on the MCC metric with five decimals.

Model	t	ACC	precision	recall	F1	MCC	prevalence
L_1	89.00	0.89	0.26	0.51	0.35	0.32	0.06
L_2	89.00	0.88	0.23	0.45	0.31	0.27	0.06
L_3	90.00	0.90	0.26	0.46	0.33	0.30	0.06
L_4	90.00	0.89	0.25	0.45	0.32	0.28	0.06
L_5	89.00	0.89	0.25	0.48	0.33	0.29	0.06
L_6	86.00	0.86	0.21	0.52	0.30	0.27	0.06
L_7	91.00	0.90	0.26	0.42	0.32	0.28	0.06
L_8	88.00	0.88	0.24	0.51	0.33	0.29	0.06
L_9	91.00	0.90	0.25	0.41	0.31	0.27	0.06
L_{10}	85.00	0.86	0.20	0.55	0.30	0.27	0.06
L_{11}	88.00	0.88	0.22	0.48	0.30	0.27	0.06
L_{12}	88.00	0.88	0.22	0.48	0.31	0.27	0.06
L_{avg}	88.67	0.88	0.24	0.48	0.32	0.28	0.06

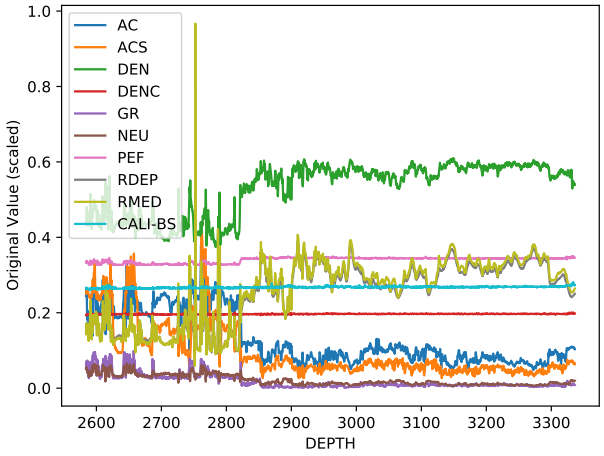
Table 5.9: LSTM results with lenient approach.

Model	t	ACC	precision	recall	F1	MCC	prevalence
L_1	93.00	0.84	0.33	0.75	0.46	0.42	0.09
L_2	93.00	0.82	0.31	0.76	0.44	0.40	0.09
L_3	95.00	0.88	0.40	0.65	0.50	0.45	0.09
L_4	94.00	0.86	0.37	0.72	0.49	0.45	0.09
L_5	95.00	0.86	0.36	0.65	0.46	0.41	0.09
L_6	92.00	0.80	0.29	0.79	0.42	0.40	0.09
L_7	95.00	0.87	0.37	0.65	0.47	0.42	0.09
L_8	95.00	0.88	0.40	0.65	0.49	0.45	0.09
L_9	94.00	0.85	0.34	0.71	0.46	0.42	0.09
L_{10}	93.00	0.84	0.32	0.76	0.45	0.42	0.09
L_{11}	93.00	0.83	0.31	0.78	0.45	0.42	0.09
L_{12}	95.00	0.86	0.34	0.62	0.44	0.39	0.09
L_{avg}	93.92	0.85	0.34	0.71	0.46	0.42	0.09

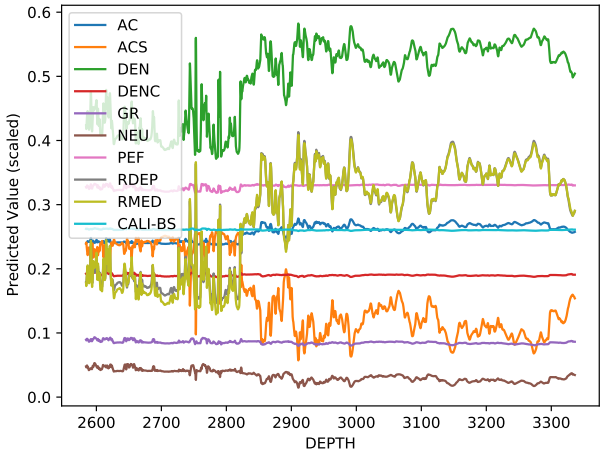
Table 5.10: LSTM results with downsampled approach.

5.3.2 Discussion

Unlike AE and VAE, the LSTM configurations have not produced any predictions of very poor quality. Figure 5.4 shows the difference between the scaled original and predicted data for configuration L_3. Some of the predicted features appear to be more accurate than others, but the configurations generally predict the data quite well. The few features with seemingly constant values in the prediction match well with the ground truth for PEF, DENC and CALI-BS. The other features follows the trend of the original data, with an acceptable amount of deviation in a few areas.



(a) Original data.



(b) Predicted scaled data (L_3).

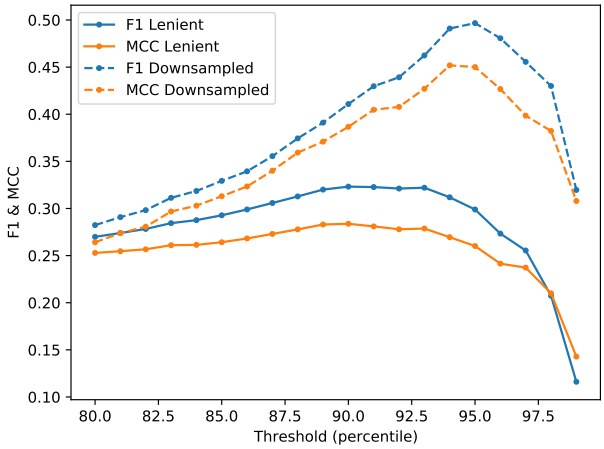
Figure 5.4: Comparison between scaled original and predicted data from L_3.

It is evident from these results that the downsampled approach performs a lot better than the lenient approach for the LSTM model. The four best performing LSTM configurations are presented in table 5.11, and are all from the downsampled model evaluation approach. The LSTM models appears to perform better as a stacked LSTM with multiple layers. The deeper LSTM layers use the output of previous LSTM layers as inputs, creating a more complex feature representation of the input. The results suggest the added complexity is beneficial for the models as they are able to capture and learn the complex relationship between features.

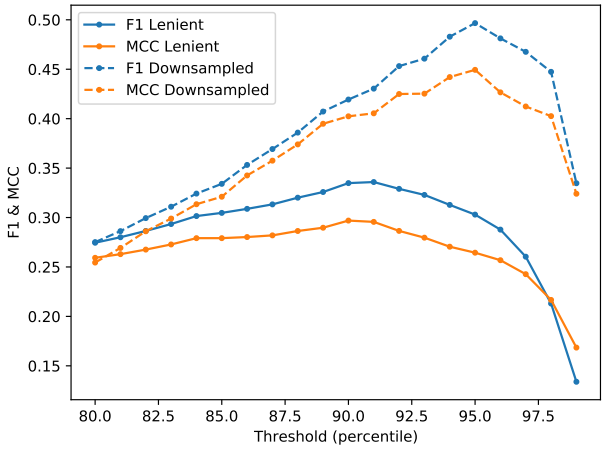
Model	t	ACC	precision	recall	F1	MCC	prevalence
L_4	94.00	0.86	0.37	0.72	0.49	0.45	0.09
L_3	95.00	0.88	0.40	0.65	0.50	0.45	0.09
L_8	95.00	0.88	0.40	0.65	0.49	0.45	0.09
L_{10}	93.00	0.84	0.32	0.76	0.45	0.42	0.09

Table 5.11: The four best performing LSTM models (downsampled).

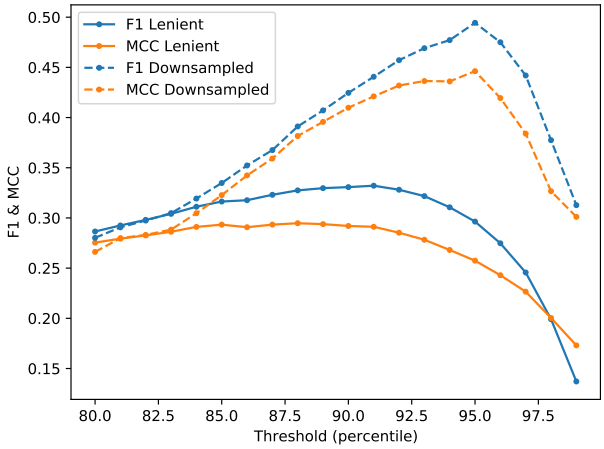
Figure 5.5 present the MCC and F1-score for the four best LSTM configurations. The metrics are recorded at different thresholds ranging from 80 to 99 with an increment of one. The lenient approach appears to generally perform worse than the downsampling approach at all thresholds. The metrics gradually increase until it reaches a peak around 89, where it starts to gradually decrease. The performance continues to decrease, and starts dropping rapidly at 97. The downsampling approach starts off with fairly poor performance, but increases quite quickly until it reaches a peak around 94. The performance of the downsampling approach starts dropping dramatically past this point.



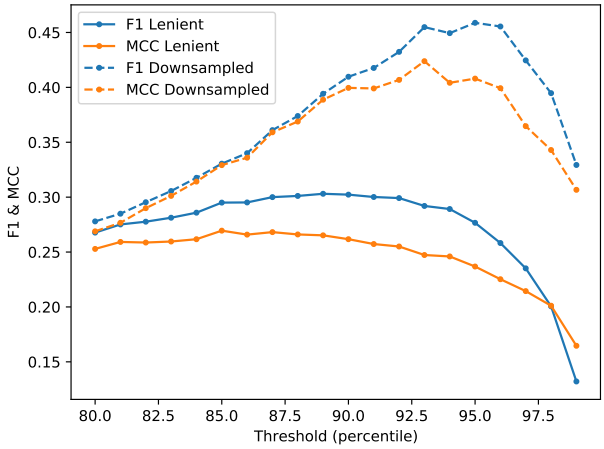
(a) Model L_4



(b) Model L_3



(c) Model L_8



(d) Model L_{10}

Figure 5.5: F1 and MCC for the best performing LSTM models.

5.4 DeepAnT

5.4.1 Results

The DeepAnT configurations are listed in table 5.12. The number of timesteps varies from 12 to 120 which corresponds to 1.2 and 12 meters respectively in the test dataset. Additionally, the timesteps are divisible by four such that the max pooling layers won't drop the last record or require padding. All configurations have a pool size of 2. The size of the dense layer is between the last pooling layer and the output layer. The kernel size is roughly between 25% and 50% of the number of timesteps. The number of filters is decided as a conceivable number of hidden features the kernels may find for the given number of timesteps. Finally, the padding method, dropout probability, optimizer, loss function, number of epochs and batch size are constant for all DeepAnT models. Causal padding is chosen such that the kernel is unable to obtain information from future timesteps. The dropout probability is set to 25%. The Adam optimizer is used with a default learning rate (0.001) and MAE is used as the loss function. During testing, DeepAnT performed best with 2 epochs and a batch size equal to the length of the current subfile.

Model	**Timesteps**	**Kernel**	**Filters**	**Dense**
D_1	12	3	7	6
D_2	12	5	7	6
D_3	12	5	11	6
D_4	32	9	21	10
D_5	32	15	21	10
D_6	32	15	27	10
D_7	60	21	27	12
D_8	60	27	27	12
D_9	60	31	27	12
D_{10}	120	31	31	20
D_{11}	120	41	31	20
D_{12}	120	49	31	20

Table 5.12: DeepAnT configurations.

Table 5.13 and 5.14 shows the results using the lenient and the downsampling method respectively. Each model configuration is shown with their best performing threshold. The downsampling method yields considerably better overall performance for all metrics except for accuracy.

Model	t	ACC	precision	recall	F1	MCC	prevalence
D_1	90.00	0.90	0.26	0.46	0.33	0.30	0.06
D_2	93.00	0.91	0.28	0.35	0.31	0.27	0.06
D_3	91.00	0.90	0.27	0.44	0.34	0.30	0.06
D_4	89.00	0.89	0.26	0.51	0.34	0.31	0.06
D_5	92.00	0.91	0.30	0.42	0.35	0.31	0.06
D_6	89.00	0.88	0.22	0.42	0.29	0.25	0.06
D_7	88.00	0.88	0.24	0.52	0.33	0.30	0.06
D_8	91.00	0.90	0.27	0.45	0.34	0.30	0.06
D_9	91.00	0.90	0.26	0.42	0.32	0.28	0.06
D_{10}	92.00	0.91	0.27	0.40	0.32	0.28	0.05
D_{11}	94.00	0.92	0.32	0.36	0.34	0.30	0.05
D_{12}	91.00	0.91	0.27	0.45	0.34	0.30	0.05
D_{avg}	90.92	0.90	0.27	0.43	0.33	0.29	0.06

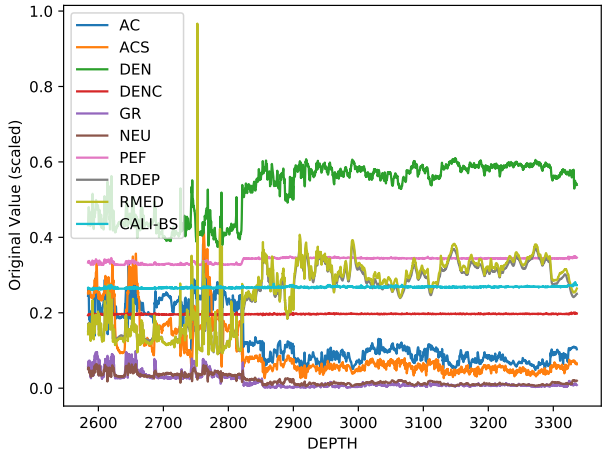
Table 5.13: DeepAnT results with lenient approach.

Model	t	ACC	precision	recall	F1	MCC	prevalence
D_1	95.00	0.88	0.41	0.72	0.52	0.48	0.09
D_2	97.00	0.90	0.45	0.47	0.46	0.40	0.09
D_3	96.00	0.89	0.44	0.65	0.53	0.48	0.09
D_4	95.00	0.88	0.41	0.69	0.52	0.48	0.09
D_5	96.00	0.90	0.46	0.65	0.54	0.49	0.09
D_6	92.00	0.81	0.29	0.76	0.42	0.39	0.09
D_7	96.00	0.90	0.44	0.63	0.52	0.47	0.09
D_8	95.00	0.88	0.41	0.67	0.51	0.46	0.09
D_9	94.00	0.87	0.37	0.72	0.49	0.45	0.09
D_{10}	95.00	0.88	0.40	0.71	0.51	0.47	0.09
D_{11}	96.00	0.90	0.43	0.63	0.51	0.47	0.09
D_{12}	96.00	0.90	0.44	0.65	0.52	0.48	0.09
D_{avg}	95.25	0.88	0.41	0.66	0.50	0.46	0.09

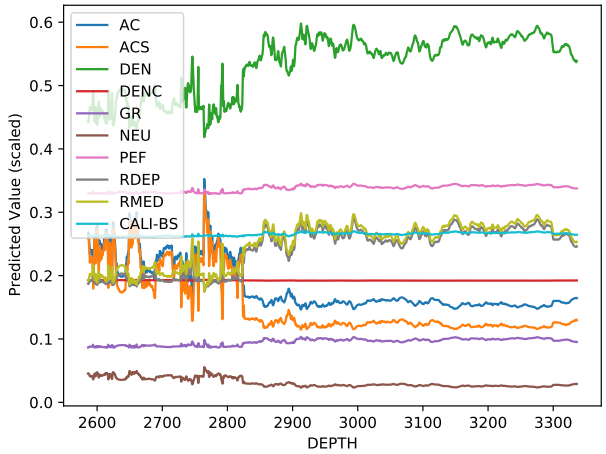
Table 5.14: DeepAnT results with downsampled approach.

5.4.2 Discussion

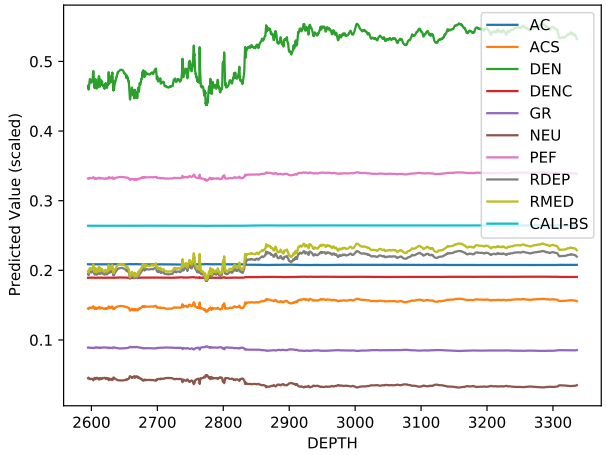
Like LSTM, the DeepAnT models generally do not suffer from poor predictions. Figure 5.6 shows a comparison of scaled actual data and predictions for configuration D_5 and D_{10}. The predictions from D_5 follow the trend in the original data with some discrepancies. The predictions from D_{10} is included in figure 5.6 to demonstrate that not all DeepAnT models make good predictions. This may stem from poor weight initialization rather than a specific hyperparameter combination.



(a) Original data.



(b) Predicted data (D_5).



(c) Predicted data (D_{10}).

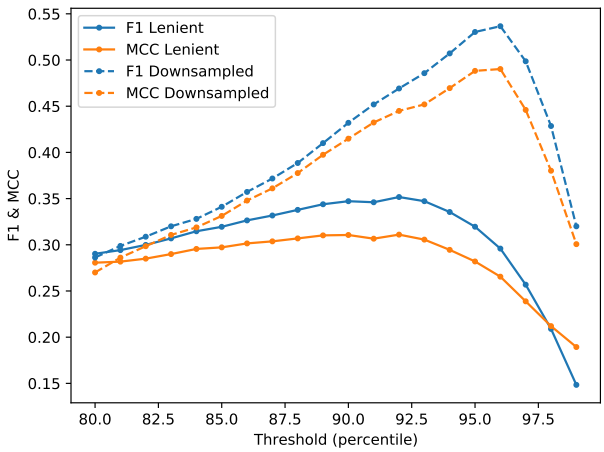
Figure 5.6: Comparison between scaled original and predicted data from D_5 and D_{10}

For all DeepAnT configurations, the downsampling method was superior compared to the lenient approach. The results from the four best models are presented in table 5.15 in descending order (based on MCC). It is evident that DeepAnT performs the best when the threshold is set to the 96th or 95th percentile. Unfortunately, it appears that no specific values of the other configurable hyperparameters are predominant based on performance. One could argue that the optimal number of timesteps is 12 because both D_1 and D_3 are among the best performing models, but D_2 is among the worst. 120 timesteps may also seem to have on average better performance, however, D_{10} suffers from making poor predictions as shown in figure 5.6c. Thus, choosing 12 or 120 timesteps cannot consistently guarantee good performance.

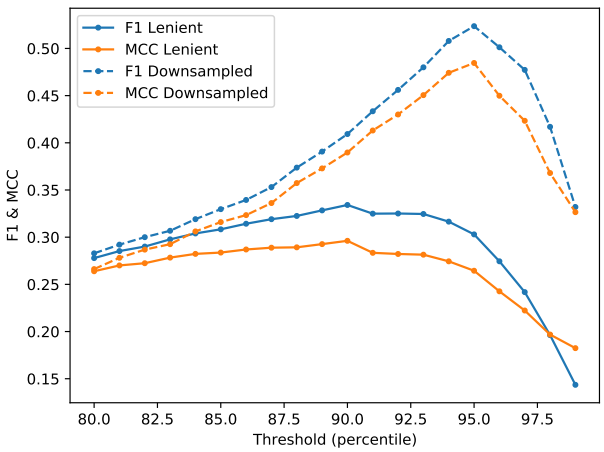
Model	t	ACC	precision	recall	F1	MCC	prevalence
D_5	96.00	0.90	0.46	0.65	0.54	0.49	0.09
D_1	95.00	0.88	0.41	0.72	0.52	0.48	0.09
D_3	96.00	0.89	0.44	0.65	0.53	0.48	0.09
D_{12}	96.00	0.90	0.44	0.65	0.52	0.48	0.09

Table 5.15: The four best performing DeepAnT models (downsampled).

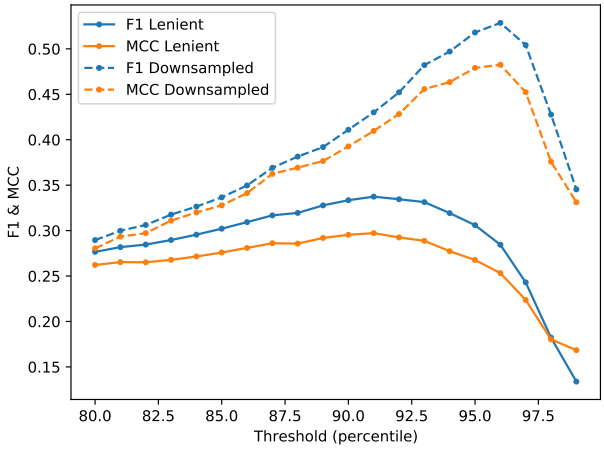
Figure 5.7 shows the F1 score and MCC for the four best DeepAnT models at different thresholds. The downsampling approach becomes advantageous for thresholds higher than the 85 percentile. The effectiveness of the lenient approach declines around the 93 threshold and rapidly declines around threshold 96. The performance of the downsampling approach plummets after the 96 threshold. This behavior likely stems from the number of false negatives increasing with the threshold, which negatively impacts MCC and recall (and therefore the F1 score).



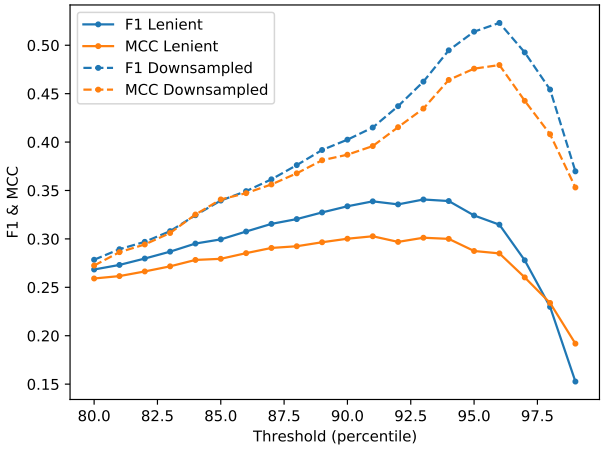
(a) Model D_5



(b) Model D_1



(c) Model D_3



(d) Model D_{12}

Figure 5.7: F1 and MCC for the best DeepAnT models.

5.5 Analysis

5.5.1 Approach Comparison

Table 5.16 shows a comparison between the top performing configuration of each model in the lenient approach. The LSTM model L_1 is our best performing model based on the MCC metric. The L_1 model also outperforms the other models in F1-score and the recall metric. Second place goes to the DeepAnT model D_5 with a slightly lower MCC score than L_1. However, D_5 outperforms the other models in prediction accuracy and the precision metric. One could argue about the correct ranking order of these two models, but our decision is based on the MCC metric which measures the overall quality of the classification. Third place goes to the autoencoder model A_{11} with the lowest MCC score of 0.31. Interestingly the A_{11} model performs very similarly to L_1 in both precision and recall metrics, but fall short in accuracy due to the lower threshold.

Model	t	ACC	precision	recall	F1	MCC	prevalence
A_{11}	88.00	0.88	0.24	0.50	0.32	0.29	0.06
L_1	89.00	0.89	0.26	0.51	0.35	0.32	0.06
D_5	92.00	0.91	0.30	0.42	0.35	0.31	0.06

Table 5.16: The best performing configuration for each model (lenient).

Table 5.17 shows a comparison between the top performing configuration of each model in the downsampling approach. The best performing model in this approach is the DeepAnT model D_5, with an MCC score of 0.49. Similarly to the results of the lenient approach, it has the highest performance in accuracy, precision, F1-score and MCC, but falls sightly short in the recall metric. Second place in this approach goes to the LSTM model L_4. This model outperforms the other models in recall, but is unable to reach the same performance as D_5 in the other metrics. The final spot goes to the autoencoder model A_7. This model has a relatively high performance in recall compared to the other models, but suffers in overall performance due to the low precision metric. The results of the downsampling approach clearly show an increase in performance. The models in this approach has a tendency to favor a lower amount of positive predictions, as seen by the increase in average threshold to roughly 95.

Model	t	ACC	precision	recall	F1	MCC	prevalence
A_7	94.00	0.84	0.32	0.70	0.44	0.40	0.09
L_4	94.00	0.86	0.37	0.72	0.49	0.45	0.09
D_5	96.00	0.90	0.46	0.65	0.54	0.49	0.09

Table 5.17: The best performing configuration for each model (downsampled).

Both the DeepAnT and LSTM model "wins" their respective model evaluation approach. However, DeepAnT is a clear winner in the downsampling approach and only slightly behind the in the lenient approach. The autoencoder is ranked third in both approaches. The VAE approach did not produce any valid models. For these reason we define our overall ranking of the models as follows:

1. DeepAnT
2. LSTM
3. AE
4. VAE

Looking at the confusion matrices in table 5.18 and the proportions of their elements in table 5.19 provides insight to the lenient and downsampling performance difference. The confusion matrices are from the lenient and downsampling approach for D_5 with a threshold of 92 and 96 respectively. In the downsampling approach, the proportion of true positives is more than double compared to the lenient method. The proportion of false positives is only slightly higher for the downsampling method. Thus, the precision is significantly higher since the proportion of true positives outweighs the proportional increase in false positives. The proportion of false negative is slightly lower for the downsampling approach. Thus, the recall sees an even greater increase. Moreover, the F1 score is increased as it is the harmonic mean of precision and recall. The increase in MCC is mainly due to the proportional change in true positives is so substantial.

	Lenient		Downsampling	
	PP	PN	PP	PN
P	447	605	110	60
N	1043	16509	130	1564

Table 5.18: Confusion matrices for D_5. The threshold is 92 and 96 for the lenient and downsampling approach respectively.

	TP	FN	FP	TN
Lenient	2.40%	3.25%	5.61%	88.74%
Downsampling	5.90%	3.22%	6.97%	83.91%

Table 5.19: The proportion of TP, FN, FP and TN for the lenient and downsampling approaches with threshold of 92 and 96 respectively.

Although the downsampling method yields higher performance metrics, it does not necessarily reflect superiority. Both approaches still make predictions in the same areas, which means the petrophysicists will be notified of the POI regardless of the evaluation approach. Thus, the additional type I and II errors close to this indication are insignificant. The downside to the lenient approach is that positive predictions tends to group around points of interest in the ground truth. This may steal the attention away from other POI that are not grouped together. A petrophysicist may potentially think that grouped POI are always better that a single one. The downsampling approach can only suggest a point of interest once every meter, which makes it less likely for many positive predictions to be grouped. Additionally, a single POI may seem more important when a whole meter is marked as potentially interesting. The downsampling may be the better approach for this reason, and not solely because of superior performance measurements. As explained in section 4.3.1, the percentiles can also be used as a score. This may prove useful for the petrophysicist as it indicates the confidence of a suggested point of interest.

5.5.2 Invalid Models & Potential Improvements

The definition of an invalid model in this project refers to models that are unable to learn from the dataset. These models typically learn to predict the average value of each feature to minimize the outcome of the loss function. This problem was present in one AE model (A_8) and in all VAE models. These models are considered invalid as they do not contribute with any useful predictions. Despite the poor quality predictions, the models tend to have a decent level of "performance" according to the evaluation metrics. The main cause of this issue is the combination of using the change in ED and percentiles to identify anomalies.

Using a threshold based on a certain percentile of the test file will always result in the same proportion of positive and negative predictions. This implies an assumption that every test file includes a certain amount of interesting points/areas. This is partly true as the large majority of well logs will contain areas that are interesting for petrophysicists. However, the proportion of interesting to non-interesting areas will not always be the same. Replacing the percentile threshold with a static threshold could potentially solve this problem, but is likely to run into two issues. If the threshold is too low, a large proportion of the test file will be marked as interesting areas and provide no benefits to the petrophysicists. If the threshold is too high, the proportion of interesting areas will be slim to none. In this scenario the petrophysicists will not be able to trust the predictions and have to manually identify the missing areas. The static threshold will probably work well on some test files, but is likely to perform very poorly on average.

The second issue is caused by the use of change in euclidean distance as the basis for interesting versus non-interesting areas. The main reason we decided to use this approach was due to different lithologies having different levels of ED. During testing we found that some lithologies have what looks like an ED "signature", where all points have almost equal ED value. This phenomenon is mainly based on the fact that the model is unable to predict all lithologies equally. When some lithologies "naturally" have a higher average ED, the model would identify the entire lithology as an anomaly with the use of a static or percentile based threshold. Consequently, areas with lower average ED would never have any of its predictions marked as an anomaly. This issue is solved by moving from ED to change in ED, as the amplitude of the euclidean distance will no longer have an impact. The

problem arises when the predictions used to create the ED curve is of poor quality. The ED values (amplitude) will generally be a lot higher for poor predictions compared to good predictions. However, since we no longer take the amplitude into account, we are unable to identify the predictions as poor. Ideally the anomaly detector module should consider both the quality of the predictions and avoid the problem associated with the ED curve at the same time.

Chapter 6

Future Work & Conclusion

6.1 Challenges

The first challenge we faced was working with a field of study we know little about. There are many aspects to learn to get a cohesive understanding of petrophysics and the work petrophysicists do. Knowing more about the drilling process helped us to understand why certain measurements were recorded. It also helped us to understand why large portions in composite logs contain missing values. Learning about basic petrophysical properties provided additional insight about the features in the dataset and why they are included. Without knowing these aspects, it would be difficult to imagine how deep learning can assist or enhance interpretation of petroleum data.

Working with the dataset itself proved to be of some challenge. Which subset of features we used heavily influenced the amount of available data. Moreover, it was important that this subset contained important features and that there were little bias towards some of the features. Furthermore, treating the data as a time series introduced some implications. Splitting a composite log into multiple separate time series meant that the model could not be trained using a standard procedure. Thankfully, the Keras API allows a model to resume training by calling the *fit* method consecutive times with different time series as input.

The most challenging part of this project was deciding, or rather, finding a suitable way to evaluate performance. Initially, we explored if different features, lithologies or lithostratigraphic units formed clusters in the latent space of a VAE. This showed some promise but had to be manually evaluated. Even if an automatic evaluation process were established, it would only be applicable for autoencoders. This study was therefore abandoned. We then came across DeepAnT whose evaluation method could also be applied to the autoencoders we had been experimenting with. Although the same evaluation method could be used for multiple model types, we could not evaluate the performance based on a confusion matrix. This sparked the idea to construct a ground truth for a single test dataset.

6.2 Future Work

There are some modifications to the four models we would like to experiment with. It would be interesting to increase the amount of input observations for AE and VAE. Instead of a single observation, the autoencoders would now accept a time series as input. The output will then be the reconstruction of the time series. With the increased input size, VAE may be able to make predictions that resemble the input and not the mean values for each corresponding feature in the training dataset. LSTM and DeepAnT can also be modified to predict multiple observations. This may be useful for real time drilling data to predict future observation several seconds or even minutes ahead of time.

Although not strictly an improvement to our implementation, studying the latent space from AE and VAE could prove valuable. We want to explore if observations of the same lithostratigraphic unit or lithology appear in clusters in the latent space. If this is true, then the clusters can be used to classify the observations' formation. Classifying the lithostratigraphic unit or lithology of an observation is exceedingly valuable as it could greatly assist boundary zonation and quality control workflows of a petrophysicist. This is perhaps more practical for VAE as its latent space is regularized.

Another improvement is to identify why a POI is interesting. It would be more helpful for a petrophysicist if a POI was labeled as "sand-to-slate", "slate-to-sand", "coal" or "hydrocarbons" to name a few examples. This

improvement somewhat coincides with classification based on clusters in a latent space but is not limited to autoencoders.

In the future, we would like to conduct a case study where we integrate POI with a petrophysicist's workflow. The feedback of such a study would indicate to which degree POI is useful. This may also shed some light on which aspects of our implementation require improvement.

6.3 Conclusion

This project aimed to use deep learning methodologies to assist in human interpretation of well logs. Incorporating deep learning into existing workflows have proved to be very beneficial in multiple other fields. Our collaboration with Logtek AS put us in a unique position where we had access to a large amount of high quality, structured well log data. We applied several deep learning methodologies to assist in boundary zonation and quality control workflows. With the vast amount of data we had available, we had the opportunity to implement deep learning algorithms that generalize the Norwegian continental shelf. This differs from the majority of related works, where researchers usually only have data available from one well or oil field.

We implemented four self-supervised deep learning algorithms used in related works for time series anomaly detection. The first two models, autoencoder and variational autoencoder, utilize reconstruction of data to identify anomalous datapoints. The last two models, LSTM and DeepAnT [39], use previously observed data to predict the next observation. Anomalies are identified by comparing the prediction to the actual observation.

We implement and expand on DeepAnT's anomaly detection module to identify points of interest in the well logs. The lack of labeled data made evaluating the performance and accuracy of the points of interest a difficult task. With the assistance of petrophysicists at Logtek, we created a "ground truth" of interesting points and areas in the test dataset. We developed two approaches for model evaluation. The first approach is *lenient* in that a predicted point of interest only has to be close to the ground truth. The second approach applies max pooling to downsample the predictions and ground truth.

The best performing model, based on MCC, for the lenient approach was the LSTM model. However, the DeepAnT models had on average better performance than LSTM. The best performing model for the downsampling approach was the DeepAnT model. The DeepAnT model also had the highest average MCC for this approach. In all cases, the downsampling approach outperformed the lenient approach. Overall the DeepAnT models is our best performing model. The LSTM model is not too far behind in performance, but is not able to keep up in the downsampling approach. The autoencoder model is the third best performing model and noticeable step down in performance. The final model, variational autoencoder, was not able to learn from the dataset and ended up only predicting constant values. The results of this model is therefore considered invalid.

The relatively good results suggests incorporating a deep learning algorithm can be beneficial for assisting human interpretation of well logs. The suggested points of interest may narrow down the number of areas the interpreter need to analyze in detail, effectively reducing the time required to quality control the well log. Further analysis is required in order to differentiate between different types of anomalies. The current implementation satisfies our goal of developing a hybrid human machine solution for well log interpretation. The deep learning models suggest the points of interest, while the petrophysicists interpret the data. Incorporating our approach into their workflow could provide us with valuable information of its strengths and weaknesses.

Bibliography

[1] M. Mohammadpoor and F. Torabi, "Big data analytics in oil and gas industry: An emerging trend," *Petroleum*, vol. 6, no. 4, pp. 321–328, 2020.

[2] J. Schmidt, M. R. G. Marques, S. Botti, and M. A. L. Marques, "Recent advances and applications of machine learning in solid-state materials science," *npj Computational Materials*, vol. 5, p. 83, Aug 2019.

[3] O. Elesen, D. Womack, and S. Lin, "How AI can pump new life into oilfields." URL `https://www.ibm.com/thought-leadership/institute-business-value/report/oil-gas-production-optimization`, 2020. [Accessed: Jun 12, 2021].

[4] *The Data Reservoir: How Big Data Technologies Advance Data Management and Analytics in E&P*, vol. Day 3 Thu, March 05, 2015 of *SPE Digital Energy Conference and Exhibition*, 03 2015. D031S021R001.

[5] E. Donaldson and D. Tiab, "Introduction to mineralogy," in *Petrophysics: Theory and Practice of Measuring Reservoir Rock and Fluid Transport Properties*, ch. 1, p. 1, Elsevier Science, 2004.

[6] Norwegian Petroleum Directorate, "Guidelines for reporting well data to authorities after completion." URL `https://www.npd.no/globalassets/1-npd/regelverk/forskrifter/en/b_og_b_digital_rapportering_e.pdf`, 2020. [Accessed: April 14, 2021].

[7] Norwegian Petroleum Directorate, "The Norwegian National Data Repository for Petroleum data - About us." URL `https://www.npd.no/en/diskos/About/`, 2021. [Accessed: Jun 9, 2021].

[8] Logtek, "Wellbore data quality control and reporting." URL `https://www.logtek.no/wellbore-data-quality-control`. [Accessed: June 8, 2021].

[9] D. Onalo, S. Adedigba, F. Khan, L. A. James, and S. Butt, "Data driven model for sonic well log prediction," *Journal of Petroleum Science and Engineering*, vol. 170, pp. 1022–1037, 2018.

[10] A. Sagheer and M. Kotb, "Time series forecasting of petroleum production using deep lstm recurrent networks," *Neurocomputing*, vol. 323, pp. 203–213, 2019.

[11] M. B. Valentín, C. R. Bom, J. M. Coelho, M. D. Correia, M. P. de Albuquerque, M. P. de Albuquerque, and E. L. Faria, "A deep residual convolutional neural network for automatic lithological facies identification in brazilian pre-salt oilfield wellbore image logs," *Journal of Petroleum Science and Engineering*, vol. 179, pp. 474–503, 2019.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[13] Y. Liu, S. Garg, J. Nie, Y. Zhang, Z. Xiong, J. Kang, and M. S. Hossain, "Deep anomaly detection for time-series data in industrial iot: A communication-efficient on-device federated learning approach," *IEEE Internet of Things Journal*, vol. 8, p. 6348–6358, Apr 2021.

[14] S. Maya, K. Ueno, and T. Nishikawa, "dlstm: a new approach for anomaly detection using deep learning with delayed prediction," *International Journal of Data Science and Analytics*, vol. 8, 09 2019.

[15] L. Martí, N. Sánchez-Pi, J. Molina, and A. C. Garcia, "Anomaly detection based on sensor data in petroleum industry applications," *Sensors*, vol. 15, pp. 2774–2797, 02 2015.

[16] Schlumberger, "LIS 79 DESCRIPTION REFERENCE MANUAL." URL `https://www.equinor.com/no/what-we-do/norwegian-continental-shelf-platforms/volve.html`, 2021. [Accessed: May 18, 2021].

[17] Petrotechnical Open Software Corporation, "POSC RP66 V1." URL http://w3.energistics.org/rp66/v1/Toc/main.html, 1998. [Accessed: May 18, 2021].

[18] Petroware, "JSON Well Log Format." URL https://jsonwelllogformat.org/, 2021. [Accessed: May 22, 2021].

[19] Canadian Well Logging Society, "LAS Version 2.0: A Digital Standard for Logs." URL https://www.cwls.org/wp-content/uploads/2014/09/LAS_20_Update_Jan2014.pdf, 2014. [Accessed: May 18, 2021].

[20] E. R. R. Crain, "Crain's petrophysical handbook." URL www.spec2000.net, 2019. [Accessed: May 19, 2021].

[21] J. D. Dziura, L. A. Post, Q. Zhao, Z. Fu, and P. Peduzzi, "Strategies for dealing with missing data in clinical trials: from design to analysis," *The Yale journal of biology and medicine*, vol. 86, no. 3, p. 343, 2013.

[22] S. García, J. Luengo, and F. Herrera, *Introduction*, pp. 59–90. Cham: Springer International Publishing, 2015.

[23] H. Kang, "The prevention and handling of the missing data," *Korean journal of anesthesiology*, vol. 64, no. 5, p. 402, 2013.

[24] A. Donner, "The relative effectiveness of procedures commonly used in multiple regression analysis for dealing with missing values," *The American Statistician*, vol. 36, no. 4, pp. 378–381, 1982.

[25] J. C. Jakobsen, C. Gluud, J. Wetterslev, and P. Winkel, "When and how should multiple imputation be used for handling missing data in randomised clinical trials – a practical guide with flowcharts," *BMC Medical Research Methodology*, vol. 17, p. 162, Dec 2017.

[26] P. Madley-Dowd, R. Hughes, K. Tilling, and J. Heron, "The proportion of missing data should not be used to guide decisions on multiple imputation," *Journal of Clinical Epidemiology*, vol. 110, pp. 63–73, 2019.

[27] S. Fielding, P. M. Fayers, A. McDonald, G. McPherson, M. K. Campbell, and t. R. s. group, "Simple imputation methods were inadequate for missing not at random (mnar) quality of life data," *Health and Quality of Life Outcomes*, vol. 6, Aug 2008.

[28] Codecademy, "Normalization." URL https://www.codecademy.com/articles/normalization. [Accessed: May 6, 2021].

[29] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, pp. 107–116, 04 1998.

[30] H. Lee and H. Kwon, "Going deeper with contextual cnn for hyperspectral image classification," *IEEE Transactions on Image Processing*, vol. 26, no. 10, pp. 4843–4855, 2017.

[31] S. Bai, J. Zico Kolter, and V. Koltun, "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling," *arXiv e-prints*, p. arXiv:1803.01271, Mar. 2018.

[32] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *ArXiv*, vol. abs/1207.0580, 2012.

[33] A. Zucconi, "Understanding the Technology Behind DeepFakes." URL `https://www.alanzucconi.com/2018/03/14/understanding-the-technology-behind-deepfakes/`, 2021. [Accessed: Jun 11, 2021].

[34] J. Rocca, "Understanding Variational Autoencoders (VAEs)." URL `https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`, 2019. [Accessed: Jan 9, 2021].

[35] C. Olah, "Understanding LSTM Networks." URL `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015. [Accessed: May 9, 2021].

[36] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[37] F. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with lstm," in *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, vol. 2, pp. 850–855 vol.2, 1999.

[38] F. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 3, pp. 189–194 vol.3, 2000.

[39] M. Munir, S. A. Siddiqui, A. Dengel, and S. Ahmed, "Deepant: A deep learning approach for unsupervised anomaly detection in time series," *IEEE Access*, vol. 7, pp. 1991–2005, 2019.

[40] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[41] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, 01 2020.

[42] Petroware, "Log I/O - Well log access library." URL `https://petroware.no/html/logio.html`, 2021. [Accessed: Jan 22, 2021].

[43] Petroware, "UoM - Units of measurement library." URL `https://petroware.no/html/uom.html`, 2021. [Accessed: Jan 25, 2021].

[44] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[45] Equinor, "Volve field data set download." URL `http://w3.energistics.org/LIS/lis-79.pdf`, 1986. [Accessed: Feb 12, 2021].

[46] Equinor, "Volve." URL `https://www.equinor.com/no/what-we-do/norwegian-continental-shelf-platforms/volve.html`, 2021. [Accessed: Mar 2, 2021].

[47] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in Neural Information Processing Systems*, vol. 29, Curran Associates, Inc., 2016.

[48] S. Semeniuta, A. Severyn, and E. Barth, "Recurrent dropout without memory loss," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, (Osaka, Japan), pp. 1757–1766, The COLING 2016 Organizing Committee, Dec. 2016.

List of Figures

List of Tables

List of code segments

Appendix A

Ground Truth

This appendix present points and areas from the wellbore *15/9-F-11 T2*. These points/areas were selected by the petrophysicists at Logtek and is treated as ground truth labels for our model evaluation approach in chapter 4.3.2.

From	To	Reason	From	To	Reason
2574.0	2575.7	Mudcake	4282.8	4283.6	Dense strings
2575.7	2577.5	Washout	4288.3	4289.4	Dense strings
4158.3	4159.5	Dense strings	4355.0	4357.0	Dense strings
4200.0	4201.0	Dense strings	4360.1	4361.6	Dense strings
4201.8	4202.5	Dense strings	4367.4	4369.1	Dense strings
4215.6	4216.5	Dense strings	4370.3	4371.6	Dense strings
4220.3	4221.2	Dense strings	4394.0	4395.7	Hydrocarbons
4228.9	4229.5	Dense strings	4397.6	4398.3	Hydrocarbons
4231.4	4231.9	Dense strings	4415.9	4417.9	Hydrocarbons
4232.3	4233.7	Dense strings	4446.0	4449.3	Dense strings
4235.5	4236.4	Dense strings	4453.2	4460.1	Dense strings
4257.5	4258.5	Dense strings	4461.2	4463.6	Dense strings
4265.1	4266.0	Dense strings	4497.8	4499.0	Coal
4269.6	4270.7	Dense strings	4506.4	4507.3	Coal

Table A.1: Ground truth areas.

Depth	Reason	Depth	Reason
2587.3	Slate to sand	2749.6	Slate to sand
2589.7	Sand to slate	2760.0	Sand to slate
2591.5	Slate to sand	2768.0	Slate to sand
2592.8	Sand to slate	2770.7	Sand to slate
2600.3	Slate to sand	2774.3	Slate to sand
2600.8	Sand to slate	2778.9	Sand to slate
2602.5	Slate to sand	2780.0	Slate to sand
2603.5	Sand to slate	2787.6	Sand to slate
2604.4	Slate to sand	2788.7	Slate to sand
2608.1	Sand to slate	2821.3	Sand to slate
2617.7	Slate to sand	2822.9	Slate to calcite
2618.2	Sand to slate	2853.8	Slate to sand
2624.7	Slate to sand	3986.6	Calcite to slate
2641.2	Sand to slate	4102.8	Slate to sand
2645.0	Slate to sand	4393.8	Slate to sand
2645.6	Sand to slate	4411.6	Sand to slate
2659.3	Slate to sand	4412.4	Slate to sand
2685.9	Sand to slate	4413.2	Sand to slate
2688.5	Slate to sand	4415.4	Slate to sand
2728.1	Sand to slate	4419.8	Sand to slate
2728.7	Slate to sand	4420.2	Slate to sand
2732.6	Sand to slate	4422.0	Sand to slate
2733.3	Slate to sand	4423.3	Slate to sand
2742.1	Sand to slate	4431.5	Sand to slate
2744.0	Slate to sand	4433.3	Slate to sand
2744.2	Sand to slate	4507.4	Sand to slate

Table A.2: Ground truth points.

Appendix B

Plots of original values

This appendix presents plots of the original scaled values for all subfiles. The colors represent the different features in the test dataset.

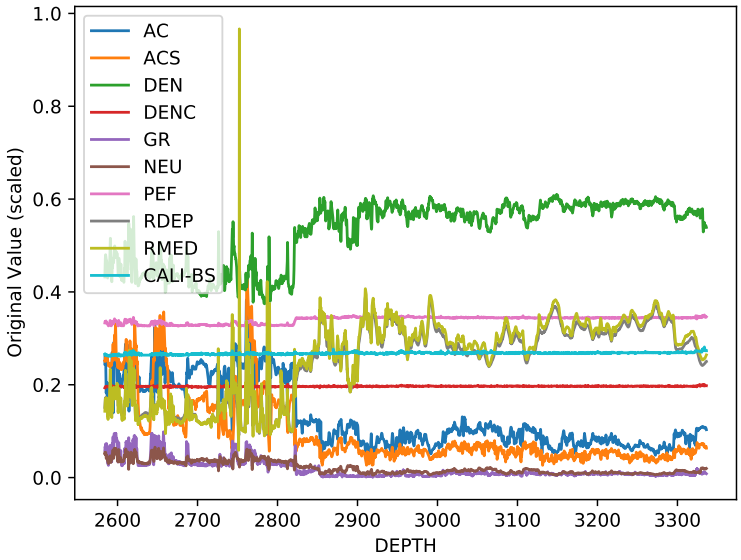


Figure B.1: Original scaled values for subfile 0.

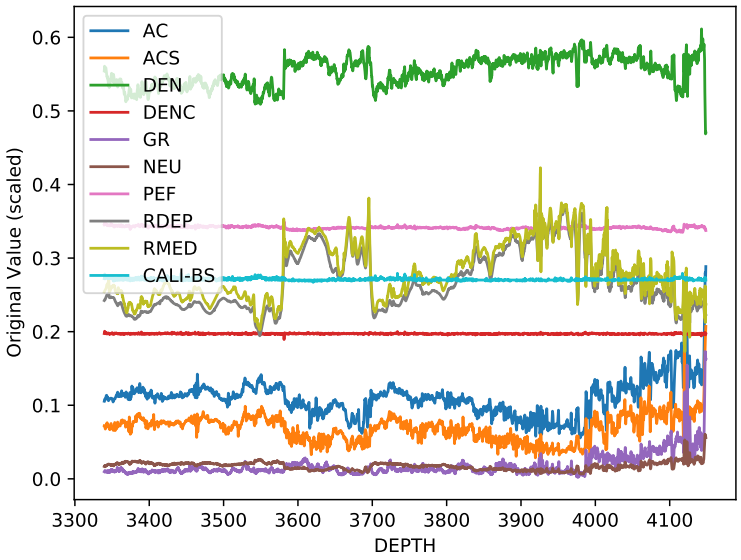


Figure B.2: Original scaled values for subfile 1.

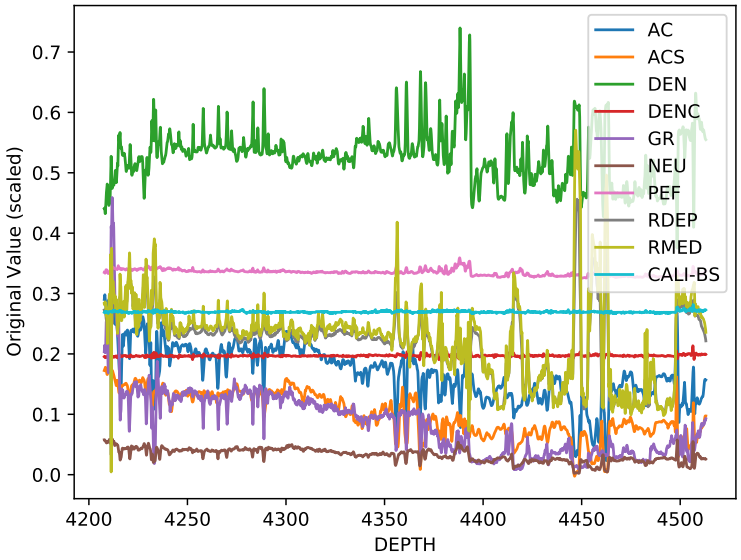
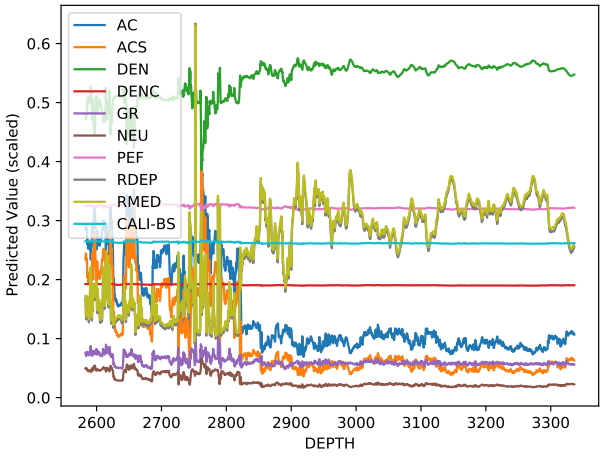


Figure B.3: Original scaled values for subfile 2.

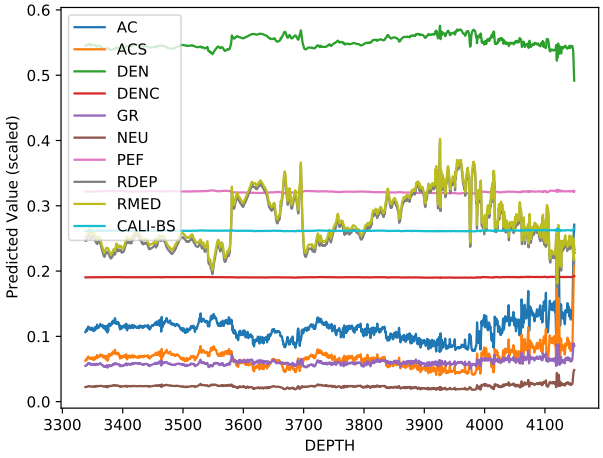
Appendix C

Predictions

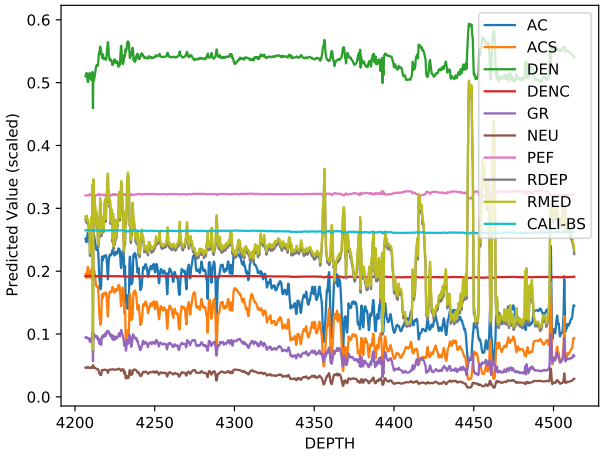
This appendix presents plots of the predicted scaled values for the best performing configuration of each model. The models presented below are A_7, V_1, L_4 and D_5. The prediction plots of all other configurations can be found on GitHub.



(a) Subfile 0

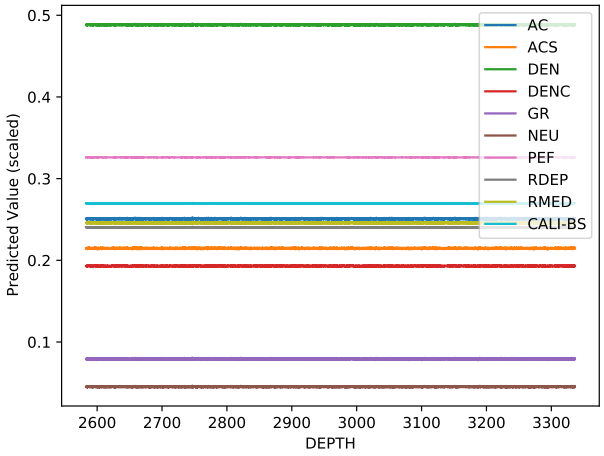


(b) Subfile 1

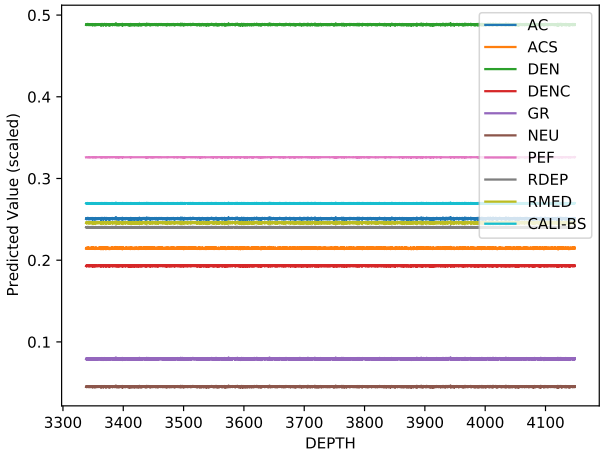


(c) Subfile 2

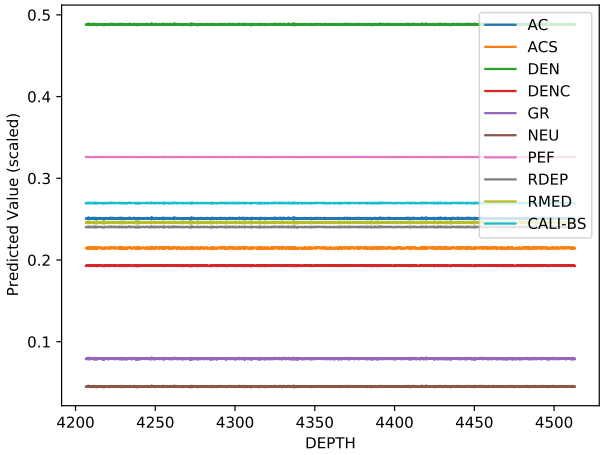
Figure C.1: AE: Predicted scaled values for all subfiles.



(a) Subfile 0

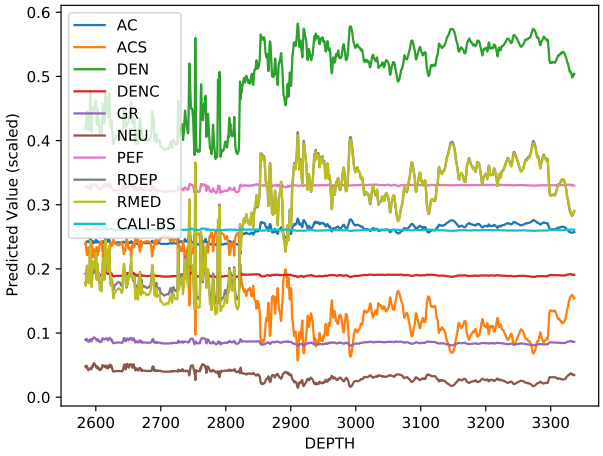


(b) Subfile 1

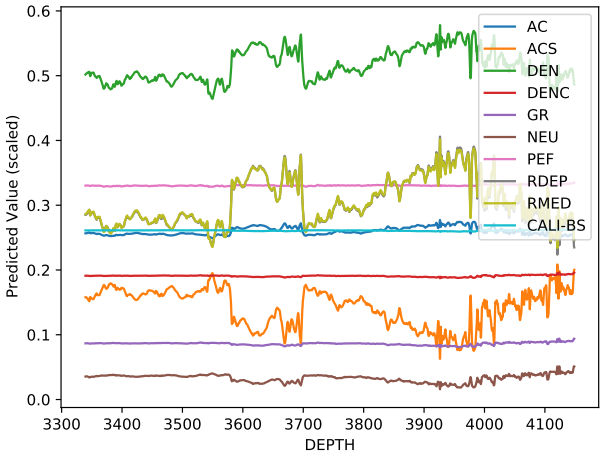


(c) Subfile 2

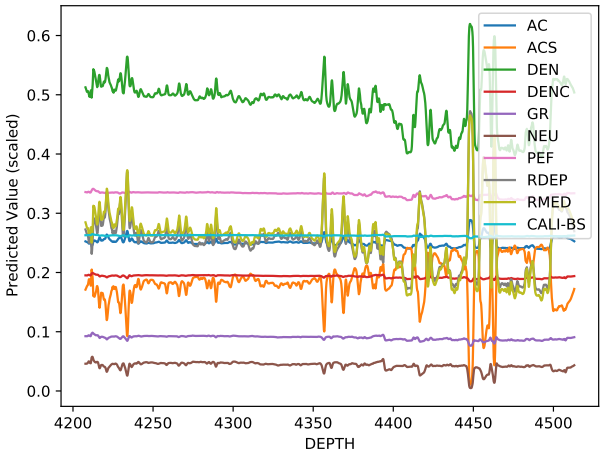
Figure C.2: VAE: Predicted scaled values for all subfiles.



(a) Subfile 0

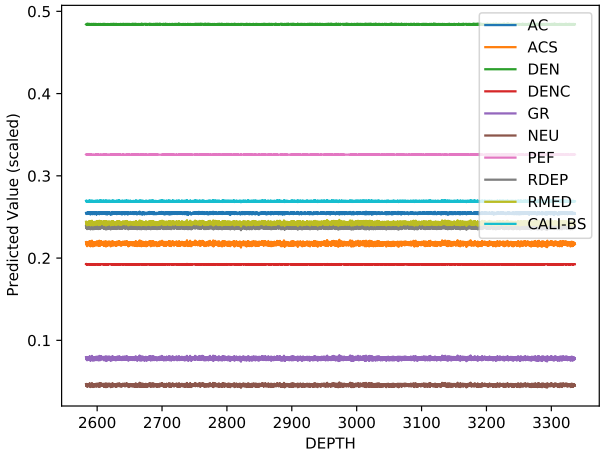


(b) Subfile 1

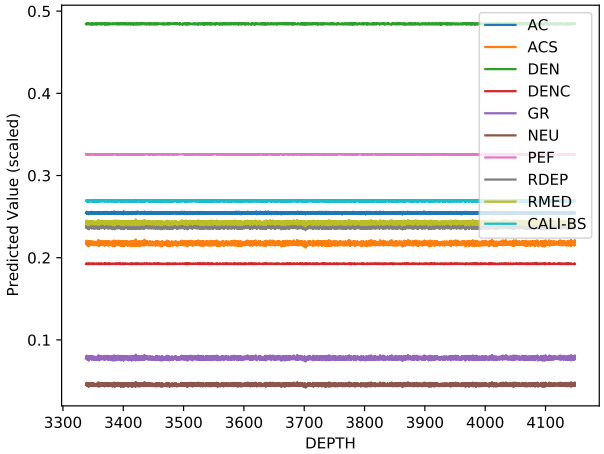


(c) Subfile 2

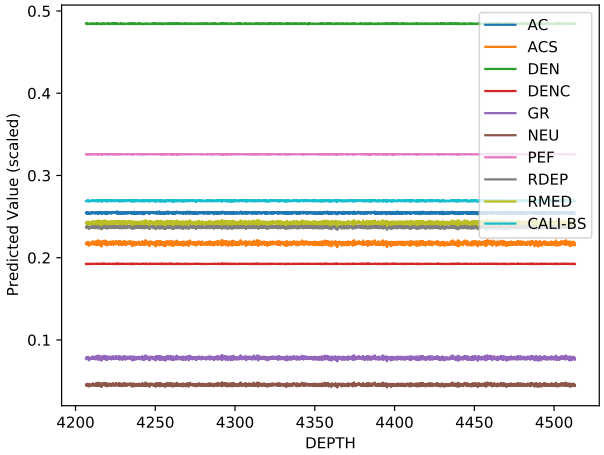
Figure C.3: LSTM: Predicted scaled values for all subfiles.



(a) Subfile 0



(b) Subfile 1



(c) Subfile 2

Figure C.4: DeepAnT: Predicted scaled values for all subfiles.

Appendix D

Euclidean Distance Plots

This appendix presents plots of the Euclidean distance for the best performing configuration of each model. The models presented below are A_7, V_1, L_4 and D_5. The Euclidean distance plots of all other configurations can be found on GitHub.

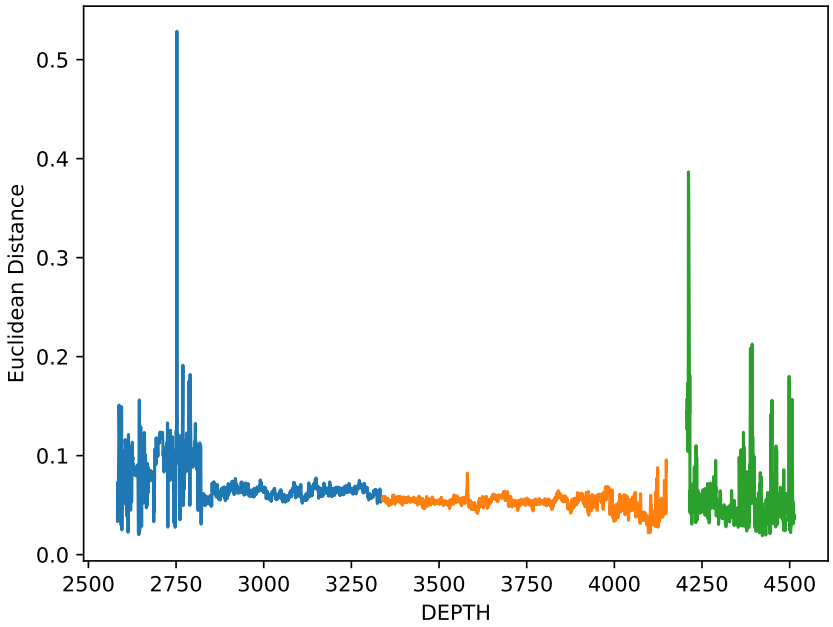


Figure D.1: Euclidean distance for A_7.

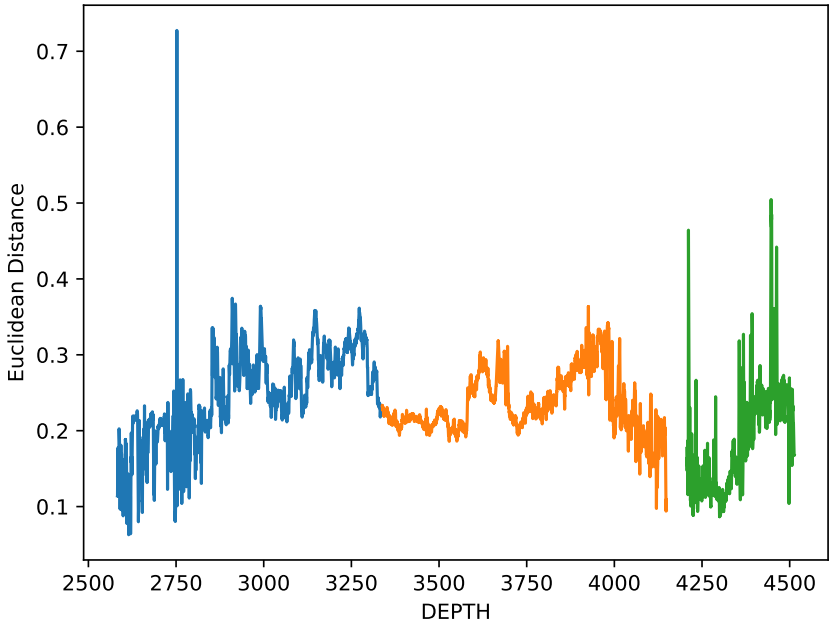


Figure D.2: Euclidean distance for V_1.

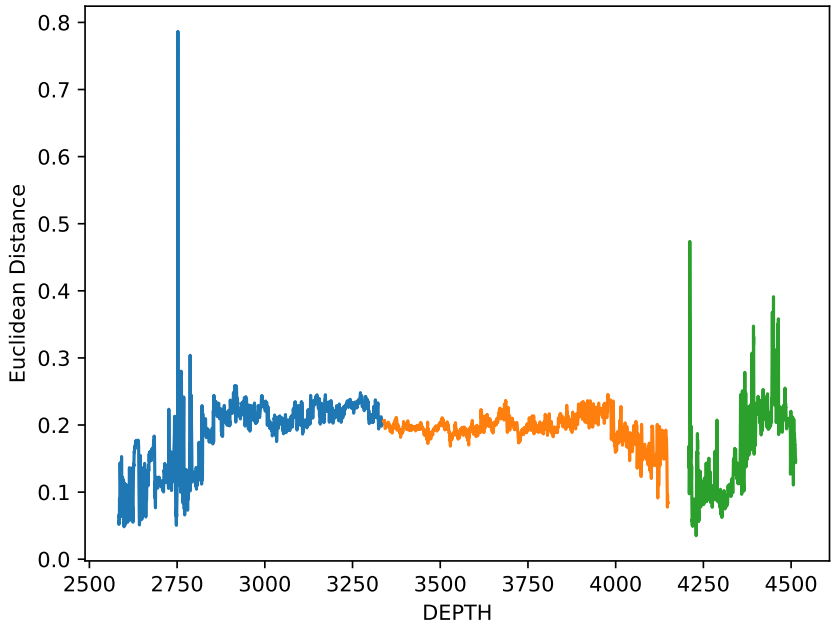


Figure D.3: Euclidean distance for L_4.

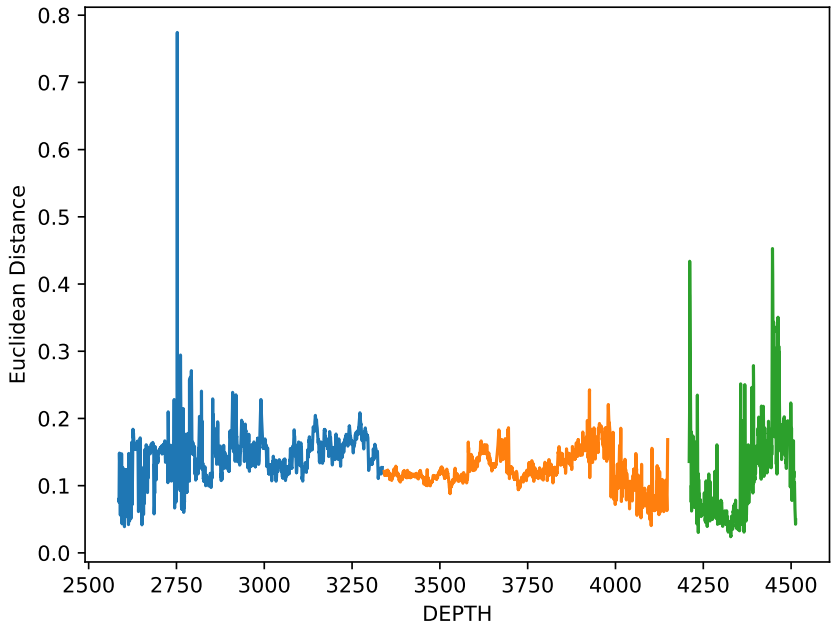


Figure D.4: Euclidean distance for D_5.

Appendix E

Change in Euclidean Distance Plots

This appendix presents plots of change in Euclidean distance for the best performing configuration of each model. The models presented below are A_7, V_1, L_4 and D_5. The change in Euclidean distance plots of all other configurations can be found on GitHub.

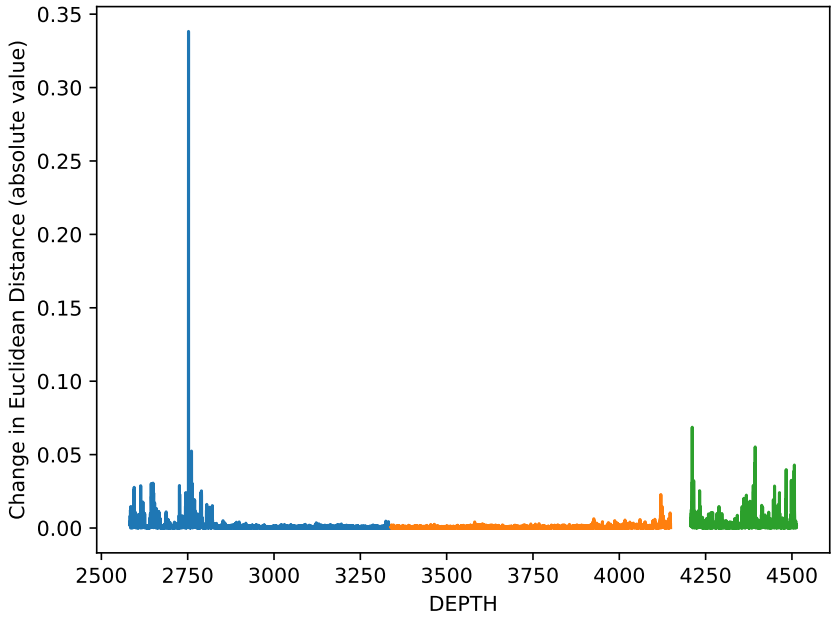


Figure E.1: Change in Euclidean distance for A_7.

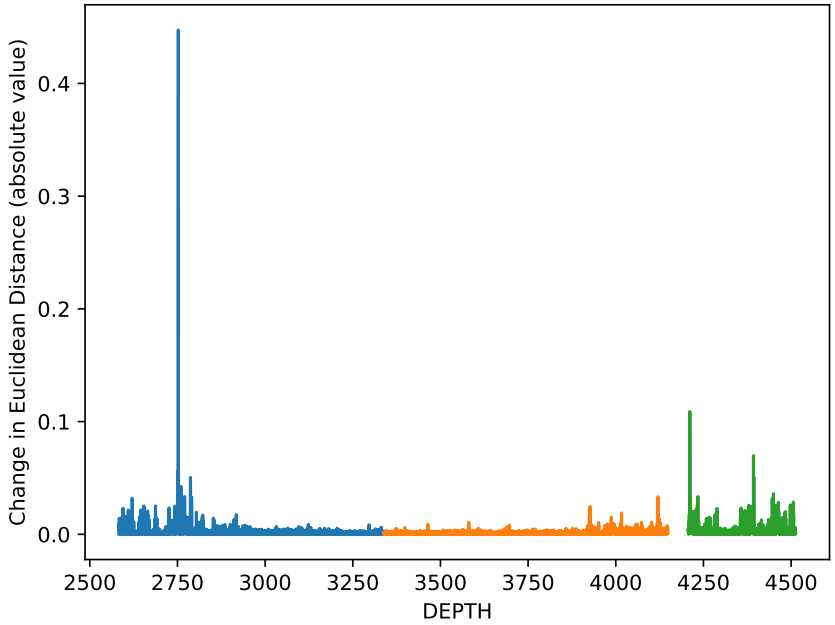


Figure E.2: Change in Euclidean distance for V_1.

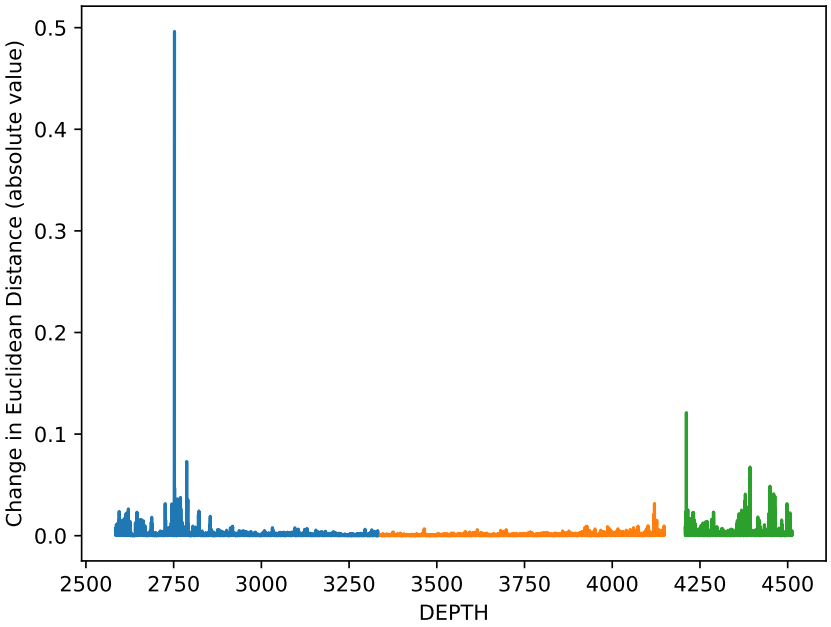


Figure E.3: Change in Euclidean distance for L_4.

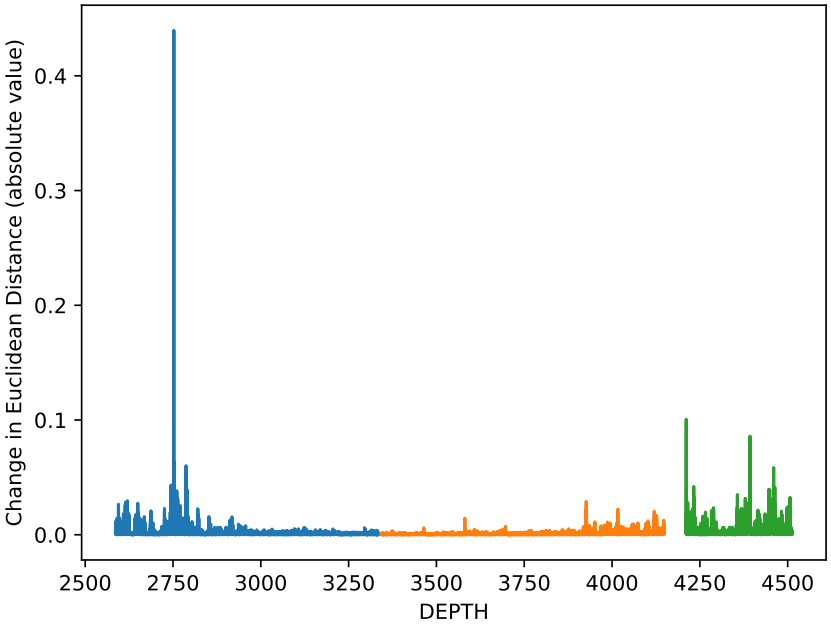


Figure E.4: Change in Euclidean distance for D_5.

Appendix F

Confusion Matrices

This appendix present the confusion matrix for the best performing configuration of each model. The models presented below are A_7, V_1, L_4 and D_5. The confusion matrix of all configurations at every threshold can be found in the associated *results.csv* file on GitHub.

	Predicted Positive (PP)	Predicted Negative (PN)
Actual Positive (P)	119	51
Actual Negative (N)	250	1453

Table F.1: Confusion matrix for A_7 (Downsampled).

	Predicted Positive (PP)	Predicted Negative (PN)
Actual Positive (P)	116	54
Actual Negative (N)	181	1522

Table F.2: Confusion matrix for V_1 (Downsampled).

	Predicted Positive (PP)	Predicted Negative (PN)
Actual Positive (P)	122	48
Actual Negative (N)	205	1495

Table F.3: Confusion matrix for L_4 (Downsampled).

	Predicted Positive (PP)	Predicted Negative (PN)
Actual Positive (P)	110	60
Actual Negative (N)	130	1564

Table F.4: Confusion matrix for D_5 (Downsampled).