Unsupervised Anomaly Detection in Multivariate Time Series Using Variational Autoencoders

Elias Aronsson

LUND
UNIVERSITY

Department of Mathematics

MSc Thesis TFRT-9999
ISSN 0280–5316

Department of Mathematics
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck.
Lund 2023

Abstract

In this master's thesis, a novel unsupervised anomaly detection tool was developed in collaboration with Sandvik Rock Processing to assist engineers and experts in analyzing large amounts of sensor data from cone crushers used in the stone crushing industry. The tool focuses on analyzing power, pressure, and CSS sensor data. A crucial preprocessing step was implemented to algorithmically identify operation segments of sufficient length, differentiating between off, idle, continuous, and discontinuous states based on power usage.

The Variational Autoencoder (VAE) employed a unique architecture with two 1D convolutions in the encoder and 1D transposed convolution in the decoder, utilizing parallel kernel sizes of 2 and 15 to capture both short-term and long-term patterns in the data. The decoder also incorporated a polynomial trend block to enhance the reconstruction. The VAE was trained on well-behaved operation segments to identify anomalous behaviour through the reconstruction error metric, Mean Absolute Percentage Error (MAPE). The anomaly detection tool achieved an F1 score of 0.89, 0.75, and 0.92 for the different sensors when tested with labelled anomalies provided by Sandvik.

Despite the challenge of limited labelled data, the tool successfully identifies the worst operation segments and can be utilized for deriving useful operation metrics. The main benefit of implementing this tool in the context of Sandvik Rock Processing's operations is the significant acceleration of sensor data analysis and the ability to highlight areas of concern for engineers and experts. Potential future improvements include using a larger dataset for training, more rigorous testing of hyperparameters, and better data collection to account for factors such as machine models and expected operating pressure and power.

Keywords: Anomaly Detection, Variational Autoencoder (VAE), Unsupervised Learning, Machine learning, AI

Acknowledgements

I want to acknowledge Dr Elisabeth Lee-Norman at Sandvik Rock Processing for supervising me throughout the project. She supplied me with the information and resources needed at every step of the way. Her patience and helpfulness, even when I had complications with getting my master's thesis started, have been very valuable to me. I am thankful for being able to do my thesis project under her. Magnus Franzen at Sandvik has collected data for me. He has also, together with Dr Elisabeth Lee- Norman categorized many rock crusher operations which were necessary for the project.

I would also like to thank my primary supervisor, Prof. Anders Heyden, for his support. He, on short notice, helped me get the thesis project off the ground.

Finally, I would like to thank my examiner Dr Alexandros Sopasakis for helping me condense and refine many diffused ideas into something more concrete at the beginning of the project.

Contents

1

Introduction

A surge of innovation in machine learning and AI has led many companies to implement some form of AI. According to research by McKinsey, AI adoption in companies has more than doubled between 2017 and 2022. In 2017, 20% of all companies had some form of AI implementation, while in 2022, 50 % of all companies have some form of AI implementation. Moreover, researchers discovered that companies, on average, increased their unique AI implementations from 1.9 in 2018 to 3.8 in 2022 [Chui et al., 2022]. An article in Forbes from 2021 claims that traditional analytic tools have served a purpose but have several shortcomings that need to be improved for today's business environment [Drai, 2021].

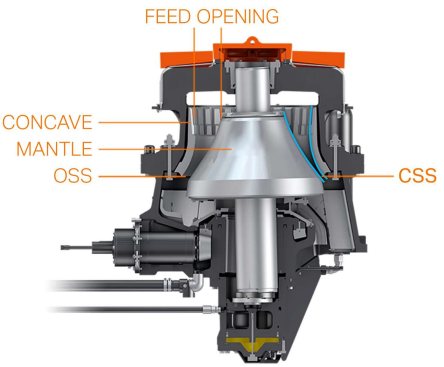
Harvard business review 2021 published an article claiming that analytical approaches using predictive models have begun to displace merely descriptive methods. The report highlights how finding the proper context to analyze data is a difficult task made much easier using AI since AI systems can determine the relevance of different facts. It is also claimed that AI-powered analytic tools can remove the need for expertise to gain insight. It is also claimed that this will allow nontechnical users to develop their analyses [Davenport and Fitts, 2021].

Sandvik Rock Processing Solutions is a part of the Sandvik group, a global engineering group providing solutions for manufacturing, mining and infrastructure industries. They are the world's leading providers of machines for rock processing industries, often selling to customers that work in either mining or construction. Sandvik manufactures and sells parts as well as equipment. They also offer maintenance and repair services and can help their customers improve the performance of their machines. In this case, understanding the function of the machines when in use by their customer is essential. When experts and engineers analyze a cone crusher's performance, the machine's sensor data is studied as a part of the analysis. One problem is that over many days or weeks of data, the task of analysis becomes very time-consuming. The vast amount of data can also raise the probability of missing important events that could be instructive in assessing the overall behaviour of the machine. This is a problem machine learning is well suited to address as AI can quickly go through large amounts of data and find strange and abnormal behaviour. AI can then guide the experts and engineers, so they can quickly find the parts of the

data that are most informative. Multiple sizes or models of Sandvik cone crushers are employed worldwide. This variance in both model and application of the cone crushers makes the construction of performance metrics difficult. During this thesis, a variational autoencoder is implemented for unsupervised anomaly detection. Variational autoencoders have been successful in similar use cases and VAEs can also be utilized in other downstream tasks.

1.1 Background

Sandvik manufactures multiple kinds of rock processing machines. During this thesis project, only data from cone crushers were treated. Cone crushers are crushers that work by squeezing the rock between an eccentrically gyrating mantle mounted to the main shaft and a concave hopper. Rocks enter the top of the crusher and are crushed as they fall through the narrow opening at the bottom.



The narrow exit through which the crushed stones fall is called the closed side setting, often called CSS [Sandvik, n.d.] The crushing occurs between the mantle and the concave. A proprietary file format called ASRI contains stored data from the cone crusher used in this thesis. The machine constantly reads multiple sensors; the results are stored in the ASRI file. Only information from the cone crusher, not the facility outside the machine, is collected. Data from three sensors were utilized, Power, Pressure and CSS. The ASRI files store data at different sampling rates depending on the data's length. All data gathered for this project was generated the week before it was retrieved.

4 States

In this thesis, the machines are defined to be in one of 4 different states:

- Off
- Idle
- Discontinuous operation
- Continuous operation

Off is defined as when no power is given to the crusher. Idle is defined as when the machine is powered and running but not crushing rocks. In this thesis, a discontinuous operation is when the machine crushes rocks for less than 20 minutes before either becoming off or idle. A continuous operation is defined as longer than 20 minutes when the machine continuously crushes rocks. The distinction between these four states is essential for two reasons. It is well known that cone crushers operate best when operating continuously for extended periods instead of with many starts and stops. Therefore, knowing how often the machines are in these four states is helpful when analyzing the performance of the cone crusher. This distinction is interesting because anomaly detection on continuous sections is a significantly easier problem than anomaly detection on all data. Since the ASRI file does not contain information on which of these states the crusher is in, an algorithm must be implemented to determine that.

1.2 Scope And Delimitations

This thesis aims to create a tool to parse long periods of sensor data effectively. The intention is to do this by making an algorithm that distinguishes between the four states the machine can be in off, idle, continuous operation and discontinuous operation. Then a VAE will be trained for unsupervised anomaly detection on the continuous operation segments. The following steps will be taken:

- Create an algorithm to determine the state of the crusher.
- Anomaly detection on continuous operations using a VAE.
- Present data to the engineers.
- Establish a good basis for future development in AI-facilitated data analysis.

Analysis and attempt to find anomalies will be exclusively on the continuous operations. Since there is an inherent instability, discontinuous operations are not indicative of the performance of the crusher. A period of rock crushing will be segmented into three sections:

- Startup.
- Steady state.
- Shutdown.

The startup in the cone crusher is when rocks are introduced into the crusher. The behaviour is unstable as the number of stones in the crusher constantly increases during this period.

The steady state is when the crusher is crushing continuously; the feed of rocks is now continuous and relatively stable. This is the state that Sandvik wants their customer's crushing processes to be in most of the time. The most optimal rock crushing occurs during the steady state, and the machine operates as intended. The rocks' feed slows down in the shutdown period and eventually stops. The dwindling feed leads to instability in the crusher's performance and behaviour.

How long the startup and end periods last is based on the cone crusher's model, the environment, and the cone crusher's application. Cone crushers are usually one of many steps in rock processing. The setup and layout of the overall process surrounding the crusher in question can affect the time it takes the machine to reach a steady state.

If the crushing period is less than 20 minutes, very little of that time is spent in a steady state. In this case, the process becomes more unpredictable, making it harder to define anomalies.

The intention is to use a VAE for anomaly detection in continuous operations and give general information about the operation time, which will be outlined in the method section. Only the power, pressure and CSS sensors will be used to construct the state algorithm and the anomaly detection.

Limiting the number of sensors provided more time to understand them and their connections to one another. This, in turn, allowed more prudent and informed choices, both in model construction and data preprocessing.

The state selection algorithm and the VAE were agnostic to the crushers' model and size. This was done due to time constraints and since it simplified data collection. There are numerous cone crusher models, and it was infeasible to gather the data set to train a model with the understanding necessary to take that into account. The state selection algorithm and the VAE are independent of the plant outside the machine.

1.3 Related Work

Anomaly detection encompasses various methodologies, including classification, distance-based, reconstruction, and clustering techniques [Schemmer et al., 2023, p. 4]. In the context of this project, a reconstruction-based approach using a Variational Autoencoder (VAE) is employed. Classification methods, such as isolation forests and support vector machines, aim to classify instances as normal or anomalous based on predefined criteria. Distance-based methods, such as k-Nearest Neighbors, measure the similarity between instances to identify anomalies. Reconstruction-based techniques, including variational autoencoders and autoencoders, reconstruct the output from lower dimensional embedding to detect anomalies. Clustering-based methods, such as K-means, Gaussian mixture models, and density-based clustering algorithms (DBSCAN), group similar instances and identify instances that deviate from these groups.

The application of Variational Autoencoders (VAEs) to unsupervised anomaly detection has emerged as a widely used technique in recent research. This approach has been successfully applied to diverse problem domains, including medical imaging, solder joints, and network intrusion detection. In medical imaging, VAEs have been utilized to identify anomalous regions within medical images, aiding in the diagnosis of diseases [Chatterjee et al., 2022; Baur et al., 2021; Zhou et al., 2020]. In the context of solder joints during mass production, VAEs have been employed to detect faulty soldering, ensuring product quality [Ulger et al., 2021]. In network intrusion detection, VAEs have been used to identify instances of network intrusion attempts, helping to enhance the security of computer networks [Fährmann et al., 2022; An and Cho, 2015].

This thesis focuses on the use of multivariate time series (MTS) data analysis to leverage VAEs for anomaly detection. A notable contribution to this field is the LSTM-VAE approach, which incorporates a VAE architecture with a Long Short-Term Memory (LSTM) backbone for temporal modelling in anomaly detection [Park et al., 2018]. By integrating LSTM layers into the VAE, the LSTM-VAE captures temporal behaviour effectively.

Another approach, called Omni-Anomaly, extends the concept of LSTM-VAE by utilizing a stochastic decoder to derive reconstruction probabilities [Su et al., 2019]. Unlike traditional reconstruction accuracy-based methods, Omni-Anomaly leverages the reconstruction probabilities to assess the likelihood of data instances being anomalous, providing a more nuanced measure of anomaly detection.

In addition, MST-VAE introduces a variational autoencoder with a multi-scale kernel, enhancing the performance of anomaly detection in MTS data [Pham et al., 2022]. The multi-scale kernel allows the VAE to capture complex temporal phenomena by combining short and long-term temporal dependencies through the use of different-sized kernels in the convolutional layers.

Time-VAE is a VAE used for interpretable lower-dimensional encoding. It incorporates time-series modelling components to introduce temporal structure and

enhance the interpretability of reconstructions [Desai et al., 2021]. It employs seasonal blocks, level blocks, scale blocks, and a residual block. The seasonal block forces the output to be a polynomial approximation by applying a Vandemonde matrix multiplication. The level block adds the same value at all time points, while the scale block multiplies all values at all times. The scale, level, and trend blocks are used in combination, and convolutional layers are employed for the residual component in reconstructions where you can track the latent codes' effect on the construction of the time series.

2

Basic AI Concepts

Artificial Intelligence (AI) refers to the development of computational systems capable of performing tasks that typically require human intelligence. It encompasses various sub-fields, including deep learning. Deep neural networks (DNNs) are artificial neural networks consisting of multiple layers of interconnected nodes called neurons. Neurons within a layer are interconnected, and each neuron receives inputs from the neurons in the previous layer. A set of learnable weights and biases transform these inputs, and each neuron applies an activation function to determine its output. How the weights and biases transform the input varies based on the type of layer used. The weights and biases are adjusted based on the training data to generalize to data outside the training set during training [Alla et al., 2019, pp. 3–6].

2.1 Activation Functions

Activation functions are mathematical operations applied to the weighted sum of inputs in a neural network's neuron. They introduce non-linearity to the network, enabling it to model complex relationships between inputs and outputs. In other words, activation functions determine the output of a neuron based on the weighted sum of its inputs. In a neural network, each neuron receives inputs from the previous layer, multiplies them by corresponding weights, and sums them up. This weighted sum is then passed through an activation function, determining the neuron's output [Alla et al., 2019, pp. 6–7].

ReLU

The rectified linear unit (ReLU) activation function is the only activation function used in this project. ReLU is one of the most commonly used activation functions and is defined as,

$$\mathrm{ReLU}(x) = \max(0, x).$$

As seen by the definition of ReLU, it leaves positive values the same while setting all negative values to 0. ReLU introduces non-linearity into the network while remaining very computationally effective compared to its alternatives [Alla et al., 2019, pp. 6–7]. Since ReLU also introduces sparsity to the network by effectively removing the negative outputs. This sparsity property encourages the network to focus on relevant and informative features by suppressing less useful or redundant ones. It can lead to more efficient representations and better generalization [Kiranyaz et al., 2021, pp. 2–3].

2.2 Layers

Fully Connected Layer

A fully connected layer is a layer in a neural network where every neuron is connected to every neuron in the previous layer. Mathematically, let's consider a fully connected layer with n neurons. Suppose the previous layer, called the input layer, has m neurons. Each neuron in the fully connected layer receives inputs from all m neurons in the input layer with associated weights. The weighted sum of these inputs is computed as,

$$z_j = \sum_{k=0}^{m} w_{(k,j)}\, x_k + b_j.$$

Here, z_j represents the weighted sum of inputs for the j-th neuron in the fully connected layer, $w_{(k,j)}$ denotes the weight connecting the k-th neuron in the input layer to the j-th neuron in the fully connected layer, x_k represents the output of the k-th neuron in the input layer and b_j is the bias term for the j-th neuron [Alla et al., 2019, pp. 5–6].

Convolutional 1D Layer

In a convolutional 1D layer, the convolution operation involves sliding a set of filters over the input data to capture local patterns and dependencies. In addition to the weights associated with each filter, biases are introduced further to enhance the flexibility and representational power of the layer. Mathematically, let's consider a convolutional 1D layer with n filters. Each filter has a width (kernel size) of K and is associated with a set of weights $\mathbf{w}_i$ and a bias term b_i. The convolution takes place with a stride of S. The convolution operation for a single filter at position j is defined as,

$$z_j = \left(\sum_{k=0}^{K-1} x_{((j \cdot S)+k)}\, w_{(j,k)} \right) + b_j.$$

Here $\mathbf{x}$ represents the input sequence, x_{j+k} denotes the $(j+k)$-th element of the input sequence, and $w_{j,k}$ represents the weight associated with the k-th element

of the filter. z_j represents the weighted sum of inputs for the j-th neuron in the convolutional 1D layer, b_j is the bias term associated with the j-th filter [Kiranyaz et al., 2021, pp. 6–7].

This equation shifts the kernel across the input sequence with a step size of S. At each position, the convolution operation is performed by element-wise multiplying the kernel with the corresponding elements of the input sequence within the kernel window, summing the results, and adding the bias term to obtain the output value z_j. Strides more significant than one reduce the output size of the layer. This is useful for reducing the spatial dimension of the data and the number of computations in the following layers. Kernel size is the filter's width, dramatically impacting what kind of patterns the layer can capture. A smaller kernel lets the network focus on capturing short-range dependencies and specific local features. A larger kernel size captures the input data's broader context and global dependencies. The choice of kernel size is a trade-off between local and global information [Pham et al., 2022, pp. 3–5, 7, 11]. Using the formulation for the convolutional 1D layer with an input of length I then, the output length is the following,

$$O = \frac{I-K}{S} + 1.$$

With a stride of 1, it might seem intuitive that the output should be the same as the input, but that is not the case [Dumoulin and Visin, 2016, p. 15]. The difference in input and output size with a stride of 1 is,

$$I - O = K - 1.$$

This can be adjusted by adding zeros at the beginning and end of the input and then shifting the convolution to account for it,

$$z_j = \left(\sum_{k=0}^{K-1} x_{\left((j \cdot S)+k-\frac{K-1}{2}\right)} \, w_{(j,k)} \right) + b_j.$$

In this equation, x_k is the k-th element in the padded array. An example of this type of 1D convolution can be seen in the figure in the Appendix.

Transposed convolutional 1D Layer

The transposed 1D convolution layer can be viewed as the inverse of the convolutional 1D layer. It can up-sample and reconstruct the input data from a lower-resolution representation. The operations in a transposed convolution are similar to that in a 1D convolution, but things like stride have a different meaning. In transposed 1D convolution, the stride decides the number of zeros to pad between the original elements of x in the padded array $\hat{x}$. Below is an example of the padded array $\hat{x}$,

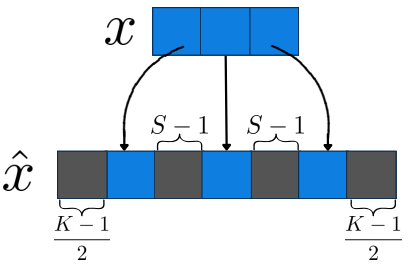


Figure 2.1 A diagram of the padding in transposed convolutional layers.

This padded array then undergoes normal convolution,

$$z_j = \left(\sum_{k=0}^{K-1} \hat{x}_{(j+k)} \, w_{(j,k)} \right) + b_j.$$

Here $\hat{\mathbf{x}}$ represents the padded input sequence, $\hat{x}_{j+k}$ denotes the $(j+k)$-th element of the padded input sequence, and $w_{(j,k)}$ represents the weight associated with the k-th element of the filter. z_j represents the weighted sum of inputs for the j-th neuron in the convolutional 1D layer, b_j is the bias term associated with the j-th filter [Dumoulin and Visin, 2016, pp. 20–22][Zeiler et al., 2010, pp. 2–3]. An example of this type of 1D transposed convolution can be seen in the figure in the Appendix.

Reshaping and Flattening Layers

Reshaping and flattening are commonly employed operations in deep neural networks that do not change the content of the data or the number of elements in the tensor but only its dimensionality. Reshaping refers to rearranging a tensor's dimensions while preserving the total number of elements. The reshaping operation involves specifying a target shape for the tensor, which defines the new arrangement of its dimensions. The target shape must be compatible with the original shape, meaning the total number of elements in the tensor remains constant. The data can

be transformed from one format to another by reshaping tensors, enabling compatibility with subsequent layers or operations. Flattening is a specific form of reshaping that collapses the dimension of the tensor. Flattening is often applied when transitioning from convolutional layers, which typically operate on spatially structured data, to fully connected layers, which require one-dimensional input.

2.3 Updating Network Weights

The loss function quantifies the discrepancy between the predicted outputs of a neural network and the desired or ground truth outputs associated with the given input data. Mathematically, let's denote the input data as $\mathbf{X}$ and the corresponding desired outputs as $\mathbf{Y}$. A deep neural network aims to approximate the actual underlying mapping between $\mathbf{X}$ and $\mathbf{Y}$. The neural network computes a set of predicted outputs, denoted as $\hat{\mathbf{Y}}$, based on the input data and the current parameters of the network.The loss function, denoted as $\mathscr{L}(\mathbf{Y}, \hat{\mathbf{Y}})$, measures the dissimilarity between the predicted outputs $\hat{\mathbf{Y}}$ and the desired outputs $\mathbf{Y}$. This discrepancy indicates the network's ability to capture the underlying patterns and generalize to unseen data. The ultimate objective during training is to minimize this loss function, thereby improving the network's predictive capabilities. The choice of a suitable loss function depends on the nature of the task. The loss function is used to quantify the error between the output and the correct output. A problem is how to adjust the network parameters best to minimize future errors. Most DNNs use backpropagation and a method based on stochastic gradient descent (SGD) for this [Kingma and Ba, 2014, p. 1].

Backpropagation

Backpropagation refers to computing gradients in a deep neural network, which is vital for updating the network's parameters. It is based on the calculus chain rule and efficiently computes the gradients by propagating the error backwards from the output layer to the input layer. To explain backpropagation, consider a deep neural network with multiple layers. Each layer consists of nodes, also known as neurons, interconnected by weighted edges. The network takes an input vector $\mathbf{X}$ and passes it forward through the layers to generate predicted outputs $\hat{\mathbf{Y}}$. During the forward pass, intermediate values, known as activations, are computed at each layer. The objective of backpropagation is to determine the impact of each weight on the overall loss function $\mathscr{L}(\mathbf{Y}, \hat{\mathbf{Y}})$. Starting at the output layer, the gradients of the loss function are computed with respect to the activations and weights. These gradients quantify the sensitivity of the loss function to changes in activations and weights. The gradients are then propagated backwards through the layers, using the chain rule, to compute the gradients at each layer. At each layer, the gradients are multiplied by the local gradients of the activation function, which captures the derivative of the activation function with respect to its inputs. This process continues

until the gradients are computed for all layers, providing the necessary information to update the network's parameters using an optimization algorithm [Goodfellow et al., 2016, pp. 197–205].

Optimization Algorithms

Stochastic Gradient Descent (SGD) is a simple optimization algorithm operating on the principle of gradient descent, but it is applied to smaller subsets of the data known as mini-batches. The key idea behind SGD is to update the model's parameters by taking small steps toward the steepest descent of the loss function with respect to the parameters, thereby finding minima in the loss function.

The primary justification for using SGD in deep learning is its computational efficiency. Training deep neural networks involves many parameters and a large quantity of training data. Computing the gradient of the loss function with respect to all the training examples at once can be computationally expensive. SGD addresses this challenge by approximating the gradient using a randomly selected mini-batch of training examples. This approximation allows for faster computation of parameter updates and facilitates efficient parameter space exploration during training.

$$\theta \leftarrow \theta - \eta \nabla \mathscr{L}_\theta(\mathbf{Y}, \hat{\mathbf{Y}})$$

Here θ represents the parameters of the model, η is the learning rate, $\mathscr{L}_\theta(\mathbf{Y}, \hat{\mathbf{Y}})$ is the loss function and $\nabla \mathscr{L}_\theta(\mathbf{Y}, \hat{\mathbf{Y}})$ is the gradient of the loss function with respect to the parameters θ. SGD allows the model to gradually converge towards a set of parameters that minimize the loss function by iteratively updating the parameters based on the gradients estimated from mini-batches.

Adaptive Moment Estimation (ADAM) is an extension of SGD that incorporates adaptive learning rates and momentum to enhance the optimization process further. It aims to address some of the limitations of SGD, such as the sensitivity to the choice of learning rate and slow convergence in some cases. The main idea behind ADAM is to maintain adaptive learning rates for each parameter based on the first and second moments of the gradients. The ADAM optimizer uses exponential moving averages of the gradient and its squared value to estimate these moments. This adaptation automatically allows ADAM to adjust the learning rates for different parameters during training.

$$
\begin{aligned}
m &\leftarrow \beta_1 \cdot m + (1-\beta_1) \cdot \nabla \mathscr{L}_\theta(\mathbf{Y}, \hat{\mathbf{Y}}) \\
v &\leftarrow \beta_2 \cdot v + (1-\beta_2) \cdot (\nabla \mathscr{L}_\theta(\mathbf{Y}, \hat{\mathbf{Y}}))^2 \\
\hat{m} &\leftarrow \frac{m}{1-\beta_1^t} \\
\hat{v} &\leftarrow \frac{v}{1-\beta_2^t} \\
\theta &\leftarrow \theta - \eta \cdot \frac{\hat{m}}{\sqrt{\hat{v}} + \varepsilon}
\end{aligned}
$$

β_1 and β_2 are hyperparameters controlling the exponential decay rates of the moving averages. Here m and v are the first and second-moment estimates, respectively, and $\hat{m}$ and $\hat{v}$ are bias-corrected moment estimates to account for the fact that the estimates are biased towards zero at the beginning of training. t is an index representing the timestep, and ε is a very small constant increasing numerical stability and avoiding division by 0. The adaptive learning rates help handle different scales of gradients across parameters, and the momentum term allows the optimizer to accelerate convergence by accumulating past gradients. The adaptive learning rates are achieved through the computation of the moving averages m and v, which are used to estimate the first and second moments of the gradients. These estimates are then bias-corrected using the terms $\hat{m}$ and $\hat{v}$. The parameters are then updated by scaling the gradients with the adaptive learning rate $\frac{\hat{m}}{\sqrt{\hat{v}}+\varepsilon}$.[Kingma and Ba, 2014, pp. 1–3]

2.4 Learning Paradigms

Three types of learning paradigms exist supervised, unsupervised, and semi-supervised.

Supervised Learning

Supervised learning is a type of machine learning where the algorithm learns from labelled examples to make predictions or classify new, unseen data. In the context of deep neural networks, supervised learning involves training a model using input-output pairs, where the inputs are the features or attributes of the data, and the outputs are the corresponding labels or target values.

In supervised learning, the training dataset consists of N labelled examples, denoted as $(\mathbf{x}_I, y_i)$, where $\mathbf{x}_i$ represents the input features and y_i represents the corresponding label or target value. The goal is to teach a function $f(\mathbf{x})$ that maps the input features to the correct output labels [Russell, 2010, pp. 528–531].

Unsupervised Learning

Unsupervised learning is a type of machine learning where the algorithm learns patterns or structures in the input data without explicit labels or target values. In unsupervised learning, the goal is to discover inherent structures, relationships, or representations within the data [Russell, 2010, pp. 528–529][Freund and Haussler, 1991, p. 1]. In deep neural networks, unsupervised learning is often used for clustering, dimensionality reduction, or generative modelling tasks. Some standard unsupervised learning algorithms include autoencoders, clustering algorithms like k-means or Gaussian mixture models, and generative adversarial networks (GANs). Unlike supervised learning, unsupervised learning does not rely on explicit labels or target values during training. Instead, it focuses on finding patterns or organizing the data to capture meaningful information without specific guidance.

Semi-supervised Learning

Semi-supervised learning is a combination of supervised and unsupervised learning. In semi-supervised learning, the algorithm is trained on a dataset containing labelled and unlabeled examples. The availability of a limited number of labelled examples and a more extensive set of unlabeled examples allows the algorithm to leverage the labelled data for supervised learning while benefiting from the additional information in the unlabeled data. In semi-supervised learning, the training dataset consists of N examples, denoted as $(\mathbf{x}_I, y_i)$ for the labelled examples and $(\mathbf{x}_j)$ for the unlabeled examples. The goal is to teach a function $f(\mathbf{x})$ that can generalize well to labelled and unlabeled examples. Semi-supervised learning algorithms typically incorporate the unsupervised learning aspect to learn valuable representations or structures from unlabeled data. These representations can then improve the model's performance on the labelled data. One common approach in deep learning is to pre-train a model using unsupervised learning (e.g., autoencoders or generative models) and then fine-tune it using the labelled data.

2.5 Feature Scaling

Feature scaling transforms data to ensure it has specific properties, often to facilitate further analysis or improve the performance of machine learning algorithms. These transformations are typically reversible, provided that information about the scaling process is retained.

Min-Max Scaling

Min-max scaling adjusts the data to fit within the range $[0,1]$. This scaling is performed using the following equation,

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}.$$

To reverse the transformation, the minimum and maximum values of the original data ($\min(x)$ and $\max(x)$) must be saved,

$$x = x'(\max(x) - \min(x)) + \min(x).$$

Z-Score Scaling

Z-score scaling is a transformation that standardizes the data by centring it around the mean (μ) and scaling it based on the standard deviation (σ). After applying this transformation, the mean of the scaled data becomes 0, and the standard deviation becomes 1,

$$x' = \frac{x - \mu}{\sigma}.$$

Here, μ represents the mean of the original data, and σ denotes its standard deviation. To invert the transformation, the mean and standard deviation values must be retained,

$$x = x'\sigma + \mu.$$

2.6 Reconstruction Metrics

There are numerous metrics for calculating the difference between a tensor and its reconstruction. In this report, the focus is on three such metrics. For the explanation of these metrics, the following notation will be used:

- x_i denotes the true value of the i^{th} element.
- $\hat{x}_i$ represents the predicted value of the i^{th} element.
- n is the number of elements.

Mean Absolute Error

Mean Absolute Error (MAE) measures the average absolute difference between predicted and true values [De Myttenaere et al., 2016, p. 4]. The formula for MAE can be expressed as,

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - \hat{x}_i|.$$

Mean Squared Error

Mean Squared Error (MSE) calculates the mean square difference between predicted and true values [Goodfellow et al., 2016, p. 105]. The formula for MSE is given by,

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2.$$

Mean Absolute Percentage Error

Mean Absolute Percentage Error (MAPE) quantifies the percentage difference between predicted and true values [De Myttenaere et al., 2016, p. 4]. The formula for MAPE can be expressed as,

$$MAPE = 100\% \times \frac{1}{n}\sum_{i=1}^{n}\left|\frac{x_i - \hat{x}_i}{x_i}\right|.$$

2.7 F_β Score

In binary classification tasks such as anomaly detection, the F_β score is a family of metrics used to evaluate the precision of a set of predictions. To define this metric, some other related terms must first be explained.

Prediction Types

For binary classification tasks, all predictions fall into one of the following categories:

- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)

In anomaly detection, TP represents correctly predicted anomalies, while TN refers to non-anomalies accurately identified as non-anomalies. FP occurs when a non-anomaly is mistakenly predicted as an anomaly, and FN occurs when an anomaly is wrongly predicted as a non-anomaly.

Recall

The recall is a metric that quantifies the proportion of actual anomalies correctly identified by the classifier. It is defined as,

$$Recall = \frac{TP}{TP + FN}.$$

Precision

Precision is a metric that measures how many elements classified as anomalies are true anomalies. It is defined as,

$$Precision = \frac{TP}{TP + FP}.$$

F_β Score

The F_β score balances recall and precision and is defined as,

$$F_\beta = \left(1 + \beta^2\right) \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall}.$$

Here, β is a constant that allows the metric to apply preferential weight to either Precision or Recall, depending on whether false negatives or false positives have more severe consequences. In this thesis, this metric will be used for the specific case $\beta = 1$, which assigns equal importance to both precision and recall [Rijsbergen C.J., 1995][Goodfellow et al., 2016, pp. 411–412],

$$F_1 = 2\frac{Precision \cdot Recall}{Precision + Recall}.$$

2.8 AUC-ROC Curve

One of the most popular and widely used evaluation metrics for classification models is the area under the receiver operating characteristic curve, abbreviated as AUC-ROC. This section explains the AUC-ROC curve, its properties, and its uses in evaluating classification models.

Receiver Operating Characteristic Curve

The Receiver Operating Characteristic (ROC) curve is a graphical representation of the performance of a binary classification model, displaying the trade-off between the true positive rate (sensitivity) and the false positive rate (1-specificity) for different classification thresholds. The true positive rate (TPR) and false positive rate (FPR) are defined as follows,

$$\text{TPR} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$
$$\text{FPR} = \frac{\text{False Positives (FP)}}{\text{False Positives (FP)} + \text{True Negatives (TN)}}.$$

The ROC curve is obtained by varying the classification threshold and plotting the TPR against the FPR. A perfect classifier would have a curve that passes through

the top-left corner of the graph, indicating a TPR of 1 and an FPR of 0. A random classifier, on the other hand, would have a diagonal line from the bottom-left corner to the top-right corner, representing a random guess.

Area Under the Curve (AUC)

The area under the ROC curve, known as the AUC, is a single scalar value that summarizes the overall performance of the classification model across all possible thresholds. The AUC ranges from 0 to 1, where a value of 1 represents a perfect classifier, and a value of 0.5 corresponds to a random classifier. The AUC can be interpreted as the probability that a randomly chosen positive instance will have a higher predicted probability than a randomly chosen negative instance. The AUC has several desirable properties that make it a popular choice for evaluating classification models:

1. **Invariance to class distribution:** The AUC is not affected by the proportion of positive and negative instances in the dataset, making it suitable for imbalanced datasets.
2. **Invariance to classification threshold:** The AUC considers the classifier's performance across all possible thresholds, comprehensively evaluating the model's performance.
3. **Rank-based metric:** The AUC is a rank-based metric, which means it only considers the order of the predicted probabilities, not their absolute values. This property makes the AUC robust to predicted probabilities' scale changes.

Computing the AUC-ROC

Several methods exist to compute the AUC, such as the trapezoidal rule and the Mann-Whitney U statistic. One popular method is the trapezoidal rule, which approximates the area under the curve by dividing it into several trapezoids and summing their areas. Given a set of FPR and TPR pairs (FPR_i, TPR_i) sorted by ascending FPR values, the AUC can be calculated as follows,

$$\text{AUC} = \sum_{i=1}^{n-1} \frac{(\text{FPR}_{i+1} - \text{FPR}_i)(\text{TPR}_{i+1} + \text{TPR}_i)}{2}.$$

Here n is the number of FPR and TPR pairs. AUC-ROC provides a comprehensive assessment of the model's performance across all classification thresholds and is robust to changes in class distribution and predicted probability scale.

3

Variational Autoencoder

The Variational Autoencoder (VAE) is a type of artificial neural network that was first introduced in 2014 by Diederik P. Kingma and Max Welling in their paper "Auto-Encoding Variational Bayes" [Kingma and Welling, 2013]. At the time, it represented a significant advancement in deep learning. The VAE quickly gained popularity in the machine learning community due to its ability to learn meaningful representations of complex data, such as images and audio. In particular, it has been used for image generation, retrieval, and manipulation tasks. Recently the VAE has been used for time series. There has been much recent work for anomaly detection in time series using semi-supervised or unsupervised learning and VAE architectures.

The VAE aims to reconstruct input data while learning a compact representation. These models are based on the concept of variational inference, and they have to balance the trade-off between the reconstruction's accuracy and the encoding's compactness. VAEs have three components, the encoder, decoder and encoding. The encoder and decoder are both deep neural networks that perform different tasks. To understand VAEs, some other concepts first need to be explained.

There is an observation x, a multivariate time series, which is the input to the VAE. In this context, the latent space, z, is the encoding of the VAE. z has a lower dimension then x and is a stochastic variable.

When making a VAE or any other encoding architecture, the goal is to make an encoder that takes the data from an observation x to an encoding z of lower dimensions and a decoder that reconstructs x from the lower dimensional encoding. This intuitively makes sense when considering that many signals have redundancies and repeatable patterns that could perfectly describe the content of a signal with less information.

3.1 Variational Inference

Variational inference is a technique for approximating a distribution by a variational distribution. The decoder is a neural network $p_\theta(x|z)$, where θ represents the parameters of the decoding network. It takes input from the latent space to the space of observations. Since the latent space is probabilistic, the mapping is also probabilistic. Since that would make the ideal encoder, knowing $p_\theta(z|x)$ is equivalent to solving the problem. When attempting to do this, problems emerge, and a technique called variational inference is employed. Variational inference is a technique to approximate complex probability distributions, such as the posterior distribution of latent variables given observed data. In the context of VAEs, there is a generative model with latent variables z and observed variables x. The true posterior distribution $p_\theta(z|x)$ can be described by the following expression,

$$p_\theta(z|x) = \frac{p_\theta(x|z)p_\theta(z)}{p_\theta(x)}.$$

Here $p_\theta(z)$ is the prior distribution for the latent space that samples from the latent space are from. $p_\theta(x)$ is the marginal likelihood. The problem is that $p_\theta(z|x)$ is often intractable due to the high-dimensional integration required to compute the marginal likelihood,

$$p_\theta(x) = \int p_\theta(x|z)p_\theta(z)dz.$$

To avoid this, a variational distribution is introduced $q_\phi(z|x)$, which is a simpler, tractable distribution that can optimize to be as close as possible to the true posterior $p_\theta(z|x)$. Meaning that the following assumption is employed,

$$q_\phi(z|x) \approx p_\theta(z|x).$$

Here $q_\phi(z|x)$ is the encoding network, where ϕ represents the parameters of the encoding network. The encoder maps the input x to a lower-dimensional representation, z. Combing the encoder and decoder, the VAE can map an observation to a latent variable z and that latent variable is then mapped to an approximation of x, $\hat{x}$,

$$x \xrightarrow{q_\phi(z|x)} z \xrightarrow{p_\theta(x|z)} \hat{x}.$$

VAE training aims to minimize the difference between the input data, x, and its reconstruction, $\hat{x}$, while ensuring that the encoding, z, is a compact data representation. Both the encoder and decoder are jointly trained using a type of stochastic gradient descent [Kingma and Welling, 2013, pp. 1–3][Bishop, 2006, pp. 462–463][Ganguly and Earp, 2021, pp. 1–2].

3.2 Kullback–Leibler Divergence

The Kullback–Leibler (KL) divergence measures the divergence or dissimilarity between two probability distributions. The KL divergence for two probability distributions P and Q of a continuous random variable x following way,

$$D_{KL}(P||Q) := \int_{\mathbb{R}} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx.$$

Here, p and q are the probability densities of the distributions P and Q. The KL divergence can be interpreted as the average difference between the logarithm of the true probability P and the logarithm of the approximating probability Q, weighted by the true probability P. This measure is asymmetric, meaning that $D_{\mathrm{KL}}(P \parallel Q)$ is generally not equal to $D_{\mathrm{KL}}(Q \parallel P)$. The KL divergence also has the property,

$$D_{KL}(P||Q) = \int_{\mathbb{R}} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx \geq 0.$$

For all probability distributions, P and Q. [Bishop, 2006, pp. 55–56]

3.3 Reparameterization Trick

Adjustments must be made to train VAEs with stochastic gradient descent (SGD) during backpropagation because z is a random variable. The trick involves transforming a random variable, z, into a deterministic transformation of another random variable, ε, and a set of parameters, μ and σ. The idea behind the reparameterization trick is to separate the stochastic component of the model from the deterministic component. In VAEs, the encoding distribution, $q_\phi(z|x)$, is typically assumed to be a Gaussian distribution with mean $\mu_\phi(x)$ and standard deviation $\sigma_\phi(x)$. Without the reparameterization trick, an overview of the VAE looks like the following.

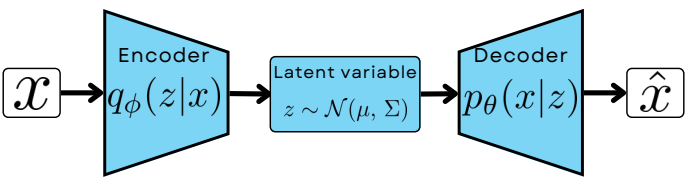


Figure 3.1 A diagram of the overview of a VAE.

In the original formulation, sampling from this distribution and computing the gradient with respect to the parameters, ϕ, would make the optimization process intractable. The reparameterization trick solves this issue by transforming the random

variable z into a deterministic transformation of another random variable ε, where ε is sampled from a standard normal distribution,

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon.$$

Here $\odot$ denotes element-wise multiplication. This transformation allows for the computation of gradients with respect to the parameters, ϕ, using standard back-propagation. The reparameterization trick enables using gradient-based optimization methods, such as stochastic gradient descent, to optimize the VAE objective [Kingma and Welling, 2013, pp. 4–5]. An overview of the VAE with the reparameterization trick is like the following.

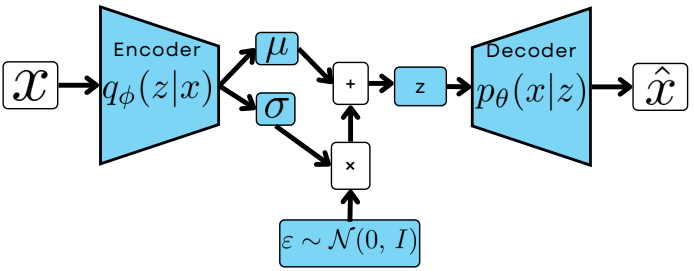


Figure 3.2 A diagram of the overview of a VAE, reparameterized.

3.4 ELBO

The evidence lower bound (ELBO) is the lower bound of the marginal log-likelihood of the observed data,

$$\log(p_\theta(x)) \geq \textbf{ELBO}.$$

This means that maximizing the ELBO corresponds to maximizing the likelihood of the data. To arrive at an expression for ELBO, KL divergence is performed on approximate posterior $q_\phi(z|x)$ and the true posterior $p_\theta(z|x)$ leading to the following expression,

$$
\begin{aligned}
D_{KL}\left(q_\phi(z|x)||p_\theta(z|x)\right) &= \mathbb{E}_{q_\phi}\left[\log\left(\frac{q_\phi(z|x)}{p_\theta(z|x)}\right)\right] \\
&= \mathbb{E}_{q_\phi}\left[\log(q_\phi(z|x))\right] - \mathbb{E}_{q_\phi}\left[\log(p_\theta(z|x))\right] \\
&= \mathbb{E}_{q_\phi}\left[\log(q_\phi(z|x))\right] - \mathbb{E}_{q_\phi}\left[\log\left(p_\theta(z,x)\right)\right] + \mathbb{E}_{q_\phi}\left[\log\left(p_\theta(x)\right)\right] \\
&= \mathbb{E}_{q_\phi}\left[\log(q_\phi(z|x))\right] - \mathbb{E}_{q_\phi}\left[\log\left(p_\theta(z,x)\right)\right] + \int q_\phi(z|x)\log(p_\theta(x))dz \\
&= \mathbb{E}_{q_\phi}\left[\log(q_\phi(z|x))\right] - \mathbb{E}_{q_\phi}\left[\log\left(p_\theta(z,x)\right)\right] + \log(p_\theta(x))\int q_\phi(z|x)dz \\
&= \mathbb{E}_{q_\phi}\left[\log(q_\phi(z|x))\right] - \mathbb{E}_{q_\phi}\left[\log\left(p_\theta(z,x)\right)\right] + \log(p_\theta(x)).
\end{aligned}
$$

This leads to this expression for KL divergence of the true and approximate posterior,

$$
D_{KL}\left(q_\phi(z|x)||p_\theta(z|x)\right) = \mathbb{E}_{q_\phi}\left[\log(q_\phi(z|x))\right] - \mathbb{E}_{q_\phi}\left[\log\left(p_\theta(z,x)\right)\right] + \log(p_\theta(x)).
$$

This expression can be rewritten to isolate the marginal log-likelihood of the observed data,

$$
\log(p_\theta(x)) = -\mathbb{E}_{q_\phi}\left[\log(q_\phi(z|x))\right] - \mathbb{E}_{q_\phi}\left[\log\left(p_\theta(z,x)\right)\right] + D_{KL}\left(q_\phi(z|x)||p_\theta(z|x)\right).
$$

Since $D_{KL}\left(q_\phi(z|x)||p_\theta(z|x)\right)$ is greater or equal to 0, the expression can be rewritten as an inequality and then simplified,

$$
\begin{aligned}
\log(p_\theta(x)) &\geq -\mathbb{E}_{q_\phi}\left[\log(q_\phi(z|x))\right] + \mathbb{E}_{q_\phi}\left[\log\left(p_\theta(z,x)\right)\right] \\
&= -\mathbb{E}_{q_\phi}\left[\log(q_\phi(z|x))\right] + \mathbb{E}_{q_\phi}\left[\log\left(p_\theta(x|z)\right)\right] + \mathbb{E}_{q_\phi}\left[\log\left(p_\theta(z)\right)\right] \\
&= \mathbb{E}_{q_\phi}\left[\log(p_\theta(x|z))\right] - \mathbb{E}_{q_\phi}\left[\log\left(\frac{q_\phi(z|x)}{p_\theta(z)}\right)\right] \\
&= \mathbb{E}_{q_\phi}\left[\log(p_\theta(x|z))\right] - D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right).
\end{aligned}
$$

This inequality for the marginal log-likelihood is ELBO. Since maximizing the right-hand side of the expression maximizes the marginal log-likelihood [Ganguly and Earp, 2021, pp. 3–7].

$$
\textbf{ELBO} = \mathbb{E}_{q_\phi}\left[\log(p_\theta(x|z))\right] - D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right).
$$

3.5 Loss Function

Since maximizing ELBO is maximizing the likelihood of the data. That means that minimizing negative ELBO also maximizes the likelihood of the data. The loss function for the VAE is defined as,

$$\mathscr{L} = -\textbf{ELBO} = -\mathbb{E}_{q_\phi}\left[\log(p_\theta(x|z))\right] + D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right).$$

This loss formulation is useful since it minimizes the reconstruction error while minimizing the error between the priors and the actual distribution. This has led to the distinction between the reconstruction loss and the KL loss,

$$\mathscr{L} = -\underbrace{\mathbb{E}_{q_\phi}\left[\log(p_\theta(x|z))\right]}_{\mathscr{L}_{Recon}} + \underbrace{D_{KL}\left(q_\phi(z|x)||p_\theta(z)\right)}_{\mathscr{L}_{KL}}.$$

KL Loss Simplification

The expression for the KL loss can be simplified further by using multivariate normal distributions with diagonal covariates as the priors for the latent space $p_\theta(z)$,

$$\begin{aligned} p_\theta(z) &= \mathscr{N}(0, I) \\ q_\phi(z|x) &= \mathscr{N}(\mu, \Sigma) \\ \textit{Where } \Sigma &= diag(\sigma_1^2, ..., \sigma_n^2). \end{aligned}$$

Using this, the expression for the KL divergence can be simplified,

$$\begin{aligned} \mathscr{L}_{KL} &= \int q_\phi(z|x) \log \frac{\frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(z-\mu)^T\Sigma^{-1}(z-\mu)\right)}{\frac{1}{(2\pi)^{n/2}} \exp\left(-\frac{1}{2}z^Tz\right)} dz \\ &= \int q_\phi(z|x)\left(-\frac{1}{2}\log|\Sigma| - \frac{1}{2}(z-\mu)^T\Sigma^{-1}(z-\mu) + \frac{1}{2}z^Tz\right)dz \\ = -\frac{1}{2}\log|\Sigma| &\int q_\phi(z|x)dz - \frac{1}{2}\int q_\phi(z|x)(z-\mu)^T\Sigma^{-1}(z-\mu)dz + \frac{1}{2}\int q_\phi(z|x)z^Tzdz. \end{aligned}$$

This expression will be broken into smaller expressions to be solved. The first integral is equal to 1 since $q_\phi(z|x)$ is a probability density function,

$$\int q_\phi(z|x)dz = 1.$$

For the second integral, the fact that the expectation of a quadratic form is given by:

$$\int q_\phi(z|x)z^Tzdz = \mathbb{E}_{q_\phi(z|x)}[z^Tz] = \mu^T\mu + \text{Tr}(\Sigma).$$

With these results, the KL divergence formula can be rewritten,

$$\mathscr{L}_{KL} = -\frac{1}{2}\log|\Sigma| - \frac{1}{2}n + \frac{1}{2}(\mu^T\mu + \mathrm{Tr}(\Sigma)).$$

Then one final simplification [Ganguly and Earp, 2021, pp. 3–7],

$$\mathscr{L}_{KL} = \frac{1}{2}\sum_{i}^{n}\left(\sigma_i^2 + \mu_i^2 - \log(\sigma_i^2) - 1\right).$$

Reconstruction Loss Simplification

Using the reparameterization trick, the latent variable z can be expressed as,

$$z = \mu(x) + \sigma(x)\cdot\varepsilon.$$

Here $\varepsilon \sim \mathscr{N}(0,I)$ is a random variable sampled from a standard normal distribution. Now, the reconstructed output of the decoder can be expressed as,

$$\hat{x} = p_\theta(x|z) = p_\theta(x|\mu(x) + \sigma(x)\cdot\varepsilon).$$

The decoder's output is deterministic and follows a Gaussian distribution with mean $\hat{x} = p_\theta(x|\mu(x)+\sigma(x)\cdot\varepsilon)$ and fixed variance σ^2. The log-likelihood of the input data x under this Gaussian distribution is given by,

$$\log p_\theta(x|z) = -\frac{1}{2\sigma^2}\|x-\hat{x}\|^2 - \frac{D}{2}\log(2\pi\sigma^2).$$

Here D is the dimensionality of the input data [Bishop, 2006, pp. 93]. Using the reparameterization trick can be used to rewrite the reconstruction error term,

$$\begin{aligned}\mathscr{L}_{Recon} &= -\mathbb{E}_{q(z|x)}\left[\log p(x|z)\right] = -\mathbb{E}_{\varepsilon\sim\mathscr{N}(0,I)}\left[\log(p_\theta(x|\mu(x)+\sigma(x)\cdot\varepsilon))\right] \\ &= \frac{1}{2\sigma^2}\mathbb{E}_{\varepsilon\sim\mathscr{N}(0,I)}\left[\|x-\hat{x}\|^2 + \frac{D}{2}\log(2\pi\sigma^2)\right].\end{aligned}$$

Since the intention is to minimize the loss, function constants are not important, so the expression can be rewritten without constants,

$$\mathscr{L}_{Recon} = \frac{1}{2\sigma^2}\mathbb{E}_{\varepsilon\sim\mathscr{N}(0,I)}\left[\|x-\hat{x}\|^2\right].$$

Using the definition of $\hat{x}$ the expression can be rewritten,

$$\mathscr{L}_{Recon} = \frac{1}{2\sigma^2}\mathbb{E}_{\varepsilon\sim\mathscr{N}(0,I)}\left[\|x-p_\theta(x|\mu(x)+\sigma(x)\cdot\varepsilon)\|^2\right].$$

The expected value is approximated by Monte Carlo sampling,

$$\mathscr{L}_{Recon} \approx \frac{1}{2N\sigma^2}\sum_{i=0}^{N}\|x-p_\theta(x|\mu(x)+\sigma(x)\cdot\varepsilon_i)\|^2.$$

Here all ε_i are random samples from $\mathcal{N}(0, I)$ [Andrieu et al., 2003]. The special case of $N = 1$ is used, which drastically simplifies the expression,

$$\mathcal{L}_{Recon} \approx \frac{1}{2\sigma^2}\|x - p_\theta(x|\mu(x) + \sigma(x) \cdot \varepsilon)\|^2 = \frac{1}{2\sigma^2}\|x - \hat{x}\|^2.$$

Here ε is a single sample from the distribution $\mathcal{N}(0, I)$. For this project, the fixed variance σ^2 can be assumed to be 1. This is because the input data is normalized using z-score scaling. With the fixed variance set to 1, the expression can be written as,

$$\mathcal{L}_{Recon} \approx \frac{1}{2}\|x - \hat{x}\|^2.$$

Final Loss Expression

Using the simplified expressions for the reconstruction loss and the KL loss and inserting them back into the expression for the loss gives the following expression,

$$\mathcal{L} = \underbrace{\frac{1}{2}\|x - \hat{x}\|^2}_{\mathcal{L}_{Recon}} + \underbrace{\frac{1}{2}\sum_i^n \left(\sigma_i^2 + \mu_i^2 - \log(\sigma_i^2) - 1\right)}_{\mathcal{L}_{KL}}.$$

This expression for the loss makes it even more clear that the reconstruction loss punishes bad reconstruction accuracy while the KL loss punishes the encoder if it produces μ and σ too far from the normal distribution.

3.6 β-VAE

A β-VAE variant of the traditional VAE incorporates an additional hyperparameter called "beta" (β) to control the trade-off between the expressiveness of the learned latent space and the reconstruction accuracy. The previous formulation of the total loss could be broken into reconstruction loss and KL loss,

$$\mathscr{L} = \mathscr{L}_{Recon} + \mathscr{L}_{KL}.$$

The adjusted loss function with β is as follows,

$$\mathscr{L} = \mathscr{L}_{Recon} + \beta \cdot \mathscr{L}_{KL}.$$

Higher values for β facilitate better disentanglement, generalization continuity and smoothness in the latent space. These desirable traits, in turn, lead to a more expressive latent space. With this implementation, it is possible to force manifold disentanglement for values of β larger than one. Values of β that are less than one lead to better reconstruction accuracy[Higgins et al., 2017][Burgess et al., 2018].

4

Data

This section will explore the format and dimension of the data processed by the model. The algorithms used to distinguish between the four states in the crusher will also be explained. The definition of an anomaly employed in this project will also be explained in detail.

4.1 Data Format

The monitoring system records successive observations that are equally spaced over time. Anomaly detection takes place for individual continuous operations. A continuous operation can be denoted as,

$$X = \{x_1, ..., x_N\},\ X \in \mathbb{R}^{N \times 3}.$$

Here N is the length of the continuous operation, three features are observed, Power, Pressure and CSS, for the length of continuous operation.

	Time ⟶						
Power	x_0^{Power}	x_1^{Power}	...	...	...	...	x_N^{Power}
Pressure	$x_0^{Pressure}$	$x_1^{Pressure}$	...	...	...	...	$x_N^{Pressure}$
CSS	x_0^{CSS}	x_1^{CSS}	...	...	...	...	x_N^{CSS}

This input has some issues since the studied continuous operations are of different lengths. A sliding window with step size one is applied to the data to account for this. The data after the sliding window has been applied to it is of the following dimensions,

$$X_{Windowed} \in \mathbb{R}^{(N-W+1) \times W \times 3}.$$

Here W is the size of the applied window. The data is constructed of N-W+1 stacked windows with three features. The shape of the data is illustrated in the following diagram,

$$\xrightarrow{\hspace{6cm}\textit{Time}\hspace{6cm}}$$

$$
\begin{array}{cccccccc}
x_0 & x_1 & x_2 & \cdots & \cdots & \cdots & \cdots & X_{W-1} \\
x_1 & x_2 & x_3 & \cdots & \cdots & \cdots & \cdots & X_W \\
x_2 & x_3 & x_4 & \cdots & \cdots & \cdots & \cdots & X_{W+1} \\
x_3 & x_4 & x_5 & \cdots & \cdots & \cdots & \cdots & X_{W+2} \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
x_{N-W+1} & x_{N-W+2} & x_{N-W+3} & \cdots & \cdots & \cdots & \cdots & X_N
\end{array}
$$

Where x_t for any valid index t,

$$x_t = \{x_t^{Power}, x_t^{Pressure}, x_t^{CSS}\}.$$

This reformatting of the data standardizes the input dimensions for the VAE. The VAE can be trained on separate windows of size W during training. The rolling window transformation is fully reversible, so the data can first be transformed, then inference can be applied to the windows then the transformation can be reversed. As seen in the table above, there are duplicates for many of the time steps x_t. Since the VAE reconstructs each window separately, this will be different in the reconstruction. The reconstruction will have the same dimension,

$$\hat{X}_{Windowed} \in \mathbb{R}^{(N-W+1)\times W\times 3}$$

Here $\hat{X}_{Windowed}$ is the windowed reconstructed produced by the VAE. An example showing the shape of the format of the reconstructed data,

$$\xrightarrow{\hspace{6cm}\textit{Time}\hspace{6cm}}$$

$$
\begin{array}{cccccccc}
x_0^0 & x_1^0 & x_2^0 & \cdots & \cdots & \cdots & \cdots & x_W^0 \\
x_1^1 & x_2^1 & x_3^1 & \cdots & \cdots & \cdots & \cdots & x_{W+1}^1 \\
x_2^2 & x_3^2 & x_4^2 & \cdots & \cdots & \cdots & \cdots & x_{W+2}^2 \\
x_3^3 & x_4^3 & x_5^3 & \cdots & \cdots & \cdots & \cdots & x_{W+3}^3 \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
\cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
x_{N-W+1}^N & x_{N-W+2}^N & x_{N-W+3}^N & \cdots & \cdots & \cdots & \cdots & X_N^N
\end{array}
$$

as seen above, x_0 gets only one approximation x_0^0. An element x_t can have between 1 and W approximations. The inverse transform was implemented such that the element x_t is the average of all approximations of x_t.

4.2 Data Flow

Each ASRI file contains data from the last week of operation. In this project, the data from the ASRI file is retrieved and stored in a CSV file using a program from Sandvik. The CSV file is then passed into the state algorithm that distinguishes which of the four states it is in. All four states are used to calculate operation metrics, but only the continuous operation segments are passed for anomaly detection. The operation metrics and the anomalies detected are then passed together as a summary. Below is a diagram of the data flow in this project.

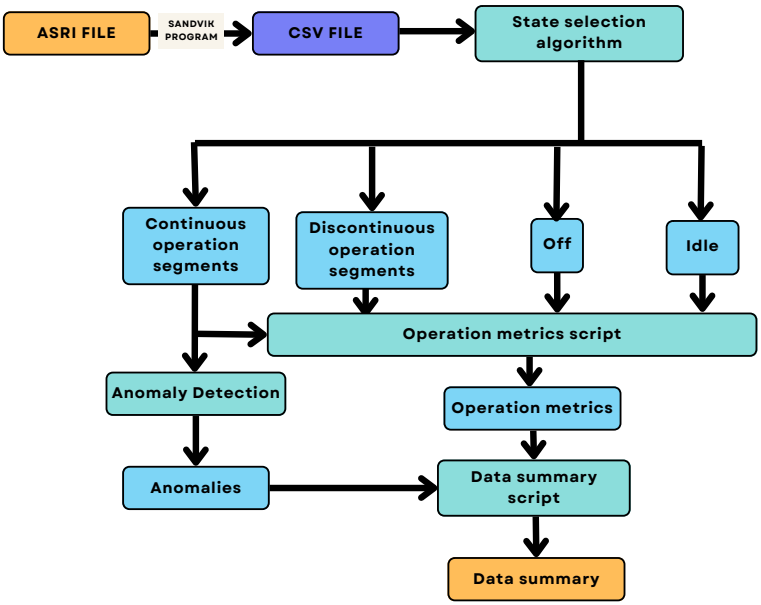


Figure 4.1 A diagram showing the data flow of the project.

4.3 Anomalies

In the context of this thesis, the focus is anomaly detection for sensor data obtained from cone crushers used for rock crushing provided by the company Sandvik. A *continuous operation* is defined as when the machine crushes rocks continuously for longer than 20 minutes. As the objects to be classified are of different lengths, a binary classification approach utilizes windows to handle this variability. Within the scope of this study, an anomaly is defined as,

> *"A continuous operation which behaves in a manner so different from a correctly operating machine that an expert or engineer would want to be notified of the operation."*

It is important to note that this definition does not necessarily imply rarity, as instances may arise where all or the majority of continuous crushing operations exhibit anomalous behaviour due to consistent mismanagement of the machinery. There are many ways to define an anomaly; even so, the definition of anomaly employed in this thesis deviates from the norm. A definition of an anomaly that more closely resembles how it is commonly used is [Aggarwal and Aggarwal, 2017],

> *"An anomaly is an observation which deviates so much from the other observations as to arouse suspicions that a different mechanism generated it."*

Additionally, the approach in this thesis departs from the standard definition in that it classifies entire operations as anomalous or non-anomalous. In contrast, other anomaly detection methods typically identify local temporal anomalies. The differences in our definition and scope make comparisons with other anomaly detection algorithms challenging. To evaluate our anomaly detection method, the reconstruction accuracy of the VAE is compared with other VAEs with similar-sized latent spaces. Performance is assessed on labelled data by calculating the F1 score, recall, precision, the Receiver Operating Characteristic (ROC) curve, and the Area Under the Curve (AUC-ROC) value.

Anomaly Score

The VAE architecture inherently learns a compressed and continuous latent space representation that captures the underlying structure and patterns within the input data. By training the VAE solely on well-behaved operation instances, the latent space is likely to primarily encode the system's normal behaviour. As a result, any deviations or anomalies introduced during testing can be detected by quantifying the reconstruction error. Well-behaved operation instances typically exhibit a certain noise level or variability due to various factors such as measurement errors or environmental conditions. Training a VAE exclusively on such instances allows it

to learn a representation accommodating this inherent noise and variability. Consequently, when faced with anomalies that deviate significantly from the learned normal behaviour, the VAE is likely to produce higher reconstruction errors, effectively indicating the presence of anomalies. The metric for reconstruction error chosen is Mean Absolute Percentage Error (MAPE).

MAPE was chosen as the reconstruction error metric because it is a scale-invariant metric that assesses the accuracy of the VAE's reconstruction independent of the magnitude or scale of the data. This characteristic is crucial when dealing with datasets collected from multiple machines of different models, as the measurements and features of interest may differ significantly in scale. By utilizing MAPE, the reconstruction error is evaluated relative to the actual values, allowing for meaningful comparison and assessment of anomalies across different machines. MAPE offers a uniform framework to assess the reconstruction accuracy of the VAE. This consistency is essential for accurately detecting anomalies and comparing the reconstruction performance across the entire dataset.

In this thesis's definition of anomaly detection, an anomaly threshold was incorporated as a crucial component. This threshold serves as a reference value, where MAPE above this threshold indicates the presence of an anomaly. A normalization approach was adopted to enhance the interpretability of the threshold values. Specifically, the respective thresholds divided the MAPE values, resulting in a ratio. Any ratio exceeding one is considered an anomaly, while ratios below 1 indicate the absence of an anomaly. The MAPE error and the anomaly thresholds for the three features can be expressed in the following way,

$$\begin{aligned} MAPE &= \begin{bmatrix} E_{Power} & E_{Pressure} & E_{CSS} \end{bmatrix} \\ Threshold &= \begin{bmatrix} t_{Power} & t_{Pressure} & t_{CSS} \end{bmatrix}. \end{aligned}$$

Using this, the notation for the anomaly score ($Anomaly_{Score}$) is as following,

$$Anomaly_{Score} = \begin{bmatrix} \frac{E_{Power}}{t_{Power}} & \frac{E_{Pressure}}{t_{Pressure}} & \frac{E_{CSS}}{t_{CSS}} \end{bmatrix}.$$

This normalization procedure aids in establishing a clear and intuitive understanding of the anomaly detection outcomes based on the relative deviation from the established thresholds.

5

VAE Architecture

The VAE consists of an encoder and a decoder, which are distinct deep neural networks trained in conjunction. The encoder maps the input data onto a lower-dimensional latent space representation through convolution operations and a fully connected layer. The decoder then reconstructs the data from the sampled latent space utilizing transposed convolution and a polynomial trend block in parallel [Oreshkin et al., 2019][Desai et al., 2021]. The encoder's convolution and the decoder's transposed convolution comprise two parallel sets of kernels with different sizes, one longer (15) and one shorter (2). Employing different kernel sizes computed in parallel enables the model to capture features spanning different scales, with large kernel sizes responsible for detecting global patterns and smaller ones for discerning local nuances [Pham et al., 2022, pp. 3–5, 7, 11].

5.1 Encoder

The encoder, comprised of convolutional layers followed by a stochastic latent space representation, efficiently captures input data's short-term and long-term features. The encoder's output contains the mean (mu) and the logarithm of the variance (logvar) of the latent space distribution, in addition to a sample drawn from the distribution itself.

Multiple convolutional operations with diverse filter numbers (80) and kernel sizes (2) are utilized for the short-term features. Each convolutional layer is followed by a ReLU activation function with the same padding that preserves the input shape. The convolutional kernels also undergo regularization with L2 weight decay. Analogously, for the long-term characteristics, a separate set of convolutional operations with an equal number of filters and padding but with a larger kernel size of 15, is employed.

The short-term and long-term feature information is integrated through an element-wise average operation performed between the two output tensors, yielding a fused representation which is then flattened into a one-dimensional tensor.

Subsequently, two fully connected layers, mu and logvar, are applied to the flattened representation within the stochastic latent space. The mu layer represents the mean of the latent distribution, whereas the logvar layer signifies the logarithm of the variance. The logarithm of the variance, used for numerical stability purposes, replaces the standard deviation. Both layers utilize an activation function of None. During the training phase, the Sampling layer takes the output tensors from the mu and logvar layers and produces a sample, serving as the stochastic latent representation. The encoder architecture is depicted in the following diagram:

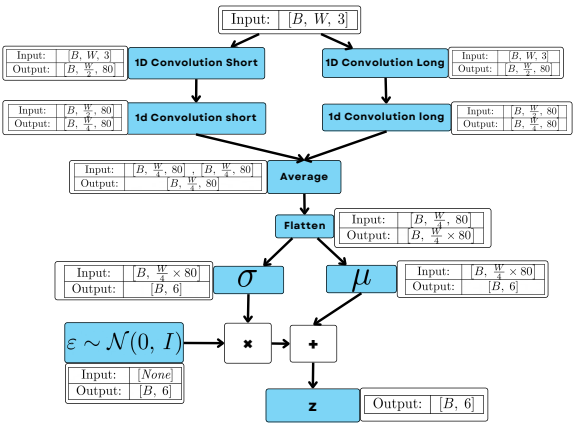


Figure 5.1 A diagram of the encoder architecture with each layer's input and output data shape.

5.2 Decoder

The decoder encompasses a series of transposed convolutional layers, fully connected layers, and a trend polynomial block. It processes the input tensor, representing the latent space, through a trend polynomial block and a combination of transposed convolution and fully connected layers. The processes are then merged through element-wise addition. The transposed convolution process mirrors that of the encoder, with two sets of transposed convolution blocks composed of different kernel sizes (2 for the shorter ones, 15 for the longer ones).

The decoder aims to reconstruct the original input data from the sampled latent vector. It first integrates trend information via a polynomial trend generation layer. Simultaneously, it applies a fully connected layer followed by a reshaping operation. The decoder executes deconvolutional layers in the reverse sequence compared to the encoder to re-establish the original feature dimensionality. Trend information is

integrated into decoding through an element-wise summation operation between the deconvolutional outputs and the trend tensor. The decoder architecture is illustrated in the diagram below:

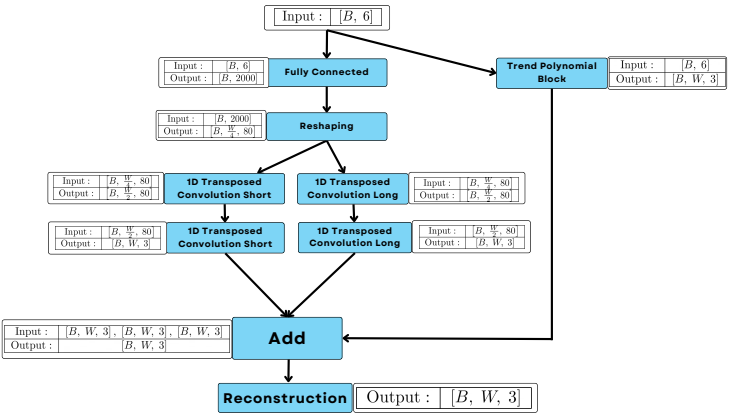


Figure 5.2 A diagram of the decoder architecture with the data shape of the input and output for each layer.

Polynomial Trend Block

The polynomial trend block is a set of layers meant to emulate polynomial approximation. First, the input data is passed through two fully connected layers and then one reshaping layer. These operations give the polynomial coefficients. The coefficients are then multiplied with a Vandermonde matrix. These operations force the block to calculate a set of coefficients to give the desired result for the polynomial. 3rd-degree polynomials were used in this case [Oreshkin et al., 2019].

6

Methodology

6.1 Establishing Anomaly Thresholds

The study engaged two domain experts from Sandvik to determine the thresholds for anomaly detection. The experts were presented with operation instances and asked to classify them as either anomalies or not using the definition of an anomaly in the context of this thesis outlined in section (4.3),

> *"A continuous operation which behaves in a manner so different from a correctly operating machine that an expert or engineer would want to be notified of the operation."*

This subjective classification approach incorporated the experts' domain knowledge and intuition. By involving multiple experts, the study aimed to capture diverse perspectives and enhance the reliability of the labelling process. Another aim of labelling the data this way was to mirror the way analysis on sensor data takes place since both experts had access to all the tools and information they usually have when evaluating the performance of a machine based on the sensor data. The labelled dataset was used to evaluate various threshold values to establish the anomaly threshold. The F1 score was chosen as the performance metric to balance minimizing false negatives and false positives. The F1 score considers precision and recall, making it suitable for evaluating the model's ability to detect anomalies effectively. The threshold value that yielded the highest F1 score was selected as the anomaly threshold for the subsequent anomaly detection system.

Furthermore, an additional set of labelled data was collected using a similar approach to evaluate the performance of the anomaly detection system. This dataset, independently labelled by the domain experts, served as a benchmark for assessing the system's effectiveness when using the established anomaly threshold.

6.2 Anomaly Detection Mechanism

The anomaly detection mechanism combines everything that has been described so far. Below is a figure showing the different operations in the anomaly detection mechanism.

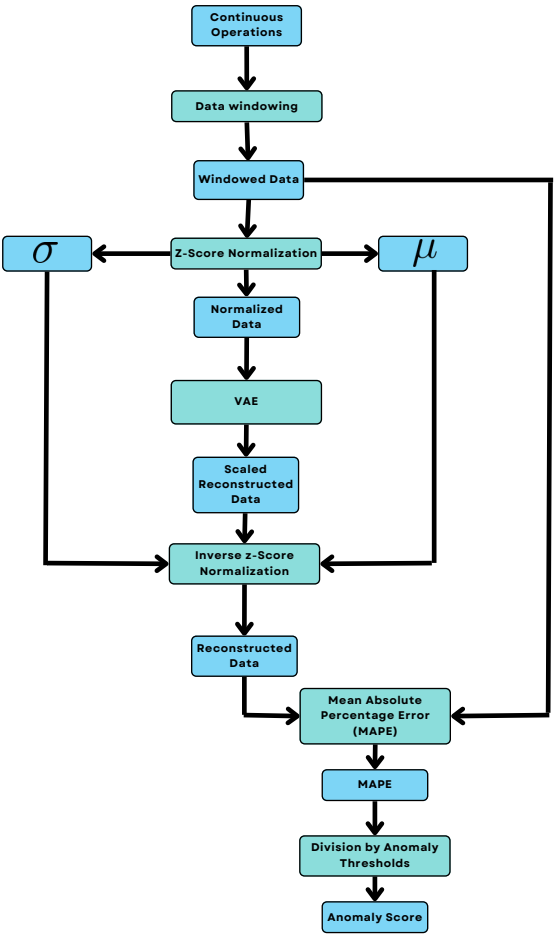


Figure 6.1 A diagram of the anomaly detection mechanism.

Firstly, the data is divided into smaller segments or windows to effectively capture local patterns and dependencies. Following windowing, a z-score normalization is applied to each windowed segment. This normalization process standardizes the

data by subtracting and dividing the mean by the standard deviation, ensuring that the data is scaled to zero mean and unit variance.

During the z-score normalization, the mean and standard deviation of the data are recorded and saved for later use. The normalized windowed data is then passed through the VAE model, consisting of an encoder and decoder network. The encoder network maps the input data to a latent representation, while the decoder network reconstructs the input from the latent space. The VAE model is trained on well-behaved time series data to learn the underlying structure and features.

Following the VAE reconstruction, an inverse z-score normalization is applied to the reconstructed data using the mean and standard deviation values saved during the normalization stage. This process restores the reconstructed data to its original scale. To quantify the discrepancy between the original and reconstructed data, the Mean Absolute Percentage Error (MAPE) is computed. The MAPE calculation compares the original windowed data and the reconstructed rescaled data. The MAPE values are divided by a pre-defined anomaly threshold to determine the anomaly score. An anomaly score greater than 1 indicates the presence of an anomaly in the corresponding windowed segment.

6.3 VAE Training

The Variational Autoencoder (VAE) training process utilized the TensorFlow framework in Python. The ADAM optimization algorithm, which combines adaptive gradient methods with momentum, was employed to update the model's weights. This algorithm has demonstrated effectiveness in optimizing neural networks.

The VAE architecture was trained using the Evidence Lower Bound (ELBO) loss function. The ELBO loss incorporates a reconstruction loss and a regularization term to balance fidelity to the input data and encourage the learning of meaningful latent representations. A beta hyperparameter of 2 was chosen for the regularization term to control the trade-off between the reconstruction loss and the regularization effect.

The available data was divided into a training set, accounting for 80% of the data, and a validation set comprising the remaining 20%. This data split facilitated the evaluation of the model's performance during training and supported hyperparameter tuning. Overfitting was mitigated by monitoring the model's performance on the validation set.

Two callback functions were employed during training to monitor and enhance the training process. The EarlyStopping callback function assessed the loss metric and halted the training if no significant improvement was observed after a specified number of epochs. This approach prevented further training when the model's performance reached a plateau, optimizing computational resources. The ReduceLROnPlateau callback function dynamically adjusted the learning rate if the loss metric ceased improving after a specific number of epochs, promoting more efficient

convergence.

The training process spanned 50 epochs, as further increasing the number of epochs did not lead to better convergence. The input data was shuffled before each epoch to enhance the model's generalization. This random shuffling was applied solely in the batch dimension, preserving the order of the features and the time dimension.

The training progress was monitored by setting the verbose flag to True, enabling the output of training updates during each epoch. This provided real-time visibility into the training process, facilitating assessment and potential troubleshooting.

Training Paradigm

The VAE was trained on a curated data set free of anomalous behaviour to learn how to represent normally operating machine behaviour accurately. This also leads to the VAE not learning the representations for anomalous data, increasing the error in reconstruction for anomalies. This is a desired property as it will help in the detection of anomalies. It is not obvious if this type of training constitutes unsupervised or semi-supervised learning.

In unsupervised learning, the goal is to extract meaningful patterns, structures, or representations from unlabelled data without any explicit information about the classes or anomalies in the dataset. The learning process relies solely on the inherent structure and distribution of the data to capture its underlying characteristics.

While the training process lacks explicit labelling or supervision, there are aspects that align with the principles of semi-supervised learning. Semi-supervised learning involves utilizing both labelled and unlabeled data to train a model. Although no explicit anomaly labels are provided, the dataset is curated to contain well-behaving data exclusively. This curation process can be seen as a form of weak supervision, where the absence of anomalies implicitly labels the data as normal.

The training of the Variational Autoencoder (VAE) using only well-behaving data can be seen as the unsupervised component of the approach. The VAE is trained to model the underlying distribution of the well-behaving data and generate accurate reconstructions. The absence of explicit anomaly labels during training aligns with the unsupervised learning paradigm.

Reconstruction accuracy as an anomaly score can be seen as introducing a level of supervision into the approach. By assuming anomalies will result in higher reconstruction errors, the approach implicitly leverages the distinction between well-behaving and poorly-behaving data. This assumption could be construed as partial supervision since it utilizes knowledge about the nature of anomalies to guide the anomaly detection process.

Since the learning in this thesis doesn't fully conform to the assumptions of semi-supervised or unsupervised learning, there is a good case for calling it either. During this thesis, the approach will be referred to as using unsupervised learning.

This will mitigate the misunderstanding that there were any labelled anomalies in the training data.

6.4 State Algorithm

The state algorithm aims to determine which of the four possible states: Off, Idle, Continuous or Discontinuous the machine is in at any time. A more detailed description of these states is in section (1.1). The algorithm only looks at the power sensor since the other sensors might be non-zero when the machine is not getting power due to measurement errors and noise.

The algorithm first isolates the segments of the run time that are where the machine is crushing stones. This is done by first setting all the values below or equal to 10 kW to zeros since none of the studied machines operate on less power. Then min-max scaling is applied to make the algorithm robust to different machines with different scales to the magnitude of power used. After this, the algorithm finds all segments where the value is more significant or equal to $threshold_1$, which is the threshold that distinguishes between crushing and non-crushing. Then the algorithm applies two thresholds $threshold_2$ and $threshold_3$. $threshold_2$ is the crushing segment length considered. Any segment shorter than that will be removed. $threshold_3$ is the minimum allowed distance between segments, and any two crushing segments closer than it will be combined into one segment. This is done to account for brief spikes of lower power values that are comparable to idle power. The pseudo-code for the part of the algorithm that distinguishes crushing and non-crushing can be seen below,

Require: fs: Sampling frequency
Require: $thresh_1, thresh_2, thresh_3$: Threshold values
Require: p, T: Power and Time arrays
  {Set all values below 10 to 0 since no machine has less power}
  $p[p \leq 10.0] \leftarrow 0.0$
  {min-max normalization to make the algorithm robust for different machine models}
  $p \leftarrow p / \max(p)$
  $ind \leftarrow$ *indices where* $p \geq thresh_1$
  $crushing \leftarrow [\,]$
  $start \leftarrow None$
  $prevEnd \leftarrow None$
  {Loops over all indices and creates an array containing the start and end of all crushing segments. It rejects all segments that are too short and combines the segments where the break in between is short enough to be assumed error.}
  for i in range($length(ind)$) **do**
    if *start is None* **then**

```
        start ← ind[i]
    else if ind[i] > ind[i−1]+1 then
        if ind[i−1]−start+1 ≥ thresh_2 then
            if (prevEnd is not None) & (start - prevEnd - 1 ≤ thresh_3) then
                {Combine segments by extending the previous segment}
                crushing[end][1] ← ind[i−1]
            else
                {Add a new segment as a separate segment}
                crushing.append([start, ind[i−1]])
                prevEnd ← ind[i−1]
            end if
            start ← ind[i]
        end if
    end if
end for
if (start is not None) & (ind[end]−start+1 ≥ thresh_2) then
    if (prevEnd is not None) and (start−prevEnd−1 ≤ thresh_3) then
        {Combine the last segment with the previous segment}
        crushing[end][1] ← ind[end]
    else
        {Add the last segment as a separate segment}
        crushing.append([start, ind[end]])
    end if
end if
```

Once all of the segments where the crushing takes place have been established, the algorithm then distinguishes between continuous and discontinuous operations based on the length of the operation. $threshold_4$ is the length needed for a crushing operation to be continuous. The non-crushing segments are then found as the segments that are not crushing. The non-rushing segments with values above $threshold_5$ are deemed idle, and those below it are considered off. This threshold is necessary as the machine can be off, but due to some small error in the sensor, it can register non-zero power. The pseudo-code for the final part of the state algorithm is below,

Require: $thresh_4, thresh_5$: Threshold values
Require: p, crushing: Power and crushing segment arrays
  $cont \leftarrow crushing[(crushing[:,1] - crushing[:,0]) \geq thresh_4, :]$
  $disc \leftarrow crushing[(crushing[:,1] - crushing[:,0]) < thresh_4, :]$
  for *segment* in *crushing* **do**
    $ind \leftarrow ind + list(range(segment[0], segment[1]+1))$
  end for
  $notCrushing \leftarrow$ *all indices from arange(p.shape[0]) not in ind*
  $offInd \leftarrow notCrushing[where\ p[notCrushing] \leq thresh_5]$
  $idleInd \leftarrow notCrushing[where\ p[notCrushing] > thresh_5]$
  $offNIdle \leftarrow [\ ]$
  for *ind* in $[offInd, idleInd]$ **do**
    $I \leftarrow (indecies\ where\ ind[1:] \neq ind[:-1] + 1) + 1$
    $I \leftarrow [0, I, ind.shape[0] - 1]$
    $segment \leftarrow ind[I]$
    $segment \leftarrow stack([segment[:-1], segment[:-1] + diff\ (I) - 1])$
    $segment[end, end] \leftarrow segment[end, end] + 1$
    $offNIdle.append(segment)$
  end for
  $off \leftarrow offNIdle[0]$
  $idle \leftarrow offNIdle[1]$

The five thresholds discussed in this section were selected based on discussions with experts at Sandvik. The figure shows an example of the algorithm's ability to distinguish between off, idle, discontinuous and continuous operations can be found in the appendix.

7

Results and Discussion

7.1 Reconstruction accuracy

To evaluate the performance of the proposed Variational Autoencoder (VAE) for unsupervised anomaly detection in multivariate time series, a comparison was conducted against two alternative models: Time-VAE and MST-VAE. The reconstruction errors were measured using three distinct metrics: Mean Absolute Percentage Error (MAPE), Mean Absolute Error (MAE), and Mean Squared Error (MSE). Table 7.1 provides a comprehensive overview of the reconstruction error comparisons,

Table 7.1 Reconstruction error on the Sandvik dataset.

VAE comparison				
Model	Sensor	Error metrics		
		MAPE	MAE	MSE
The VAE	Power	**1.65%**	**2.08**	**8.92**
	Pressure	**1.56%**	**0.02**	$\mathbf{5.55}\times10^{-4}$
	CSS	**0.12%**	**0.04**	$\mathbf{3.85}\times10^{-3}$
Time-VAE	Power	2.7%	4.17	33.25
	Pressure	3.48%	0.07	9.02×10^{-3}
	CSS	1.05%	0.24	2.22×10^{-1}
MST-VAE	Power	23.4%	32.4	1.4×10^{3}
	Pressure	32.5%	0.47	0.43
	CSS	55.1%	23.6	8.1×10^{2}

The table shows that the VAE achieved competitive performance across all the sensor types evaluated, demonstrating its effectiveness in capturing the underlying patterns and accurately reconstructing the multivariate time series data. Specifically, when considering the MAPE metric, the VAE achieved an average error of 1.65% for the Power sensor, 1.56% for the Pressure sensor, and 0.12% for the CSS sensor.

The corresponding MAE values were 2.08, 0.02, and 0.04, while the MSE values were 8.92, 5.55×10^{-4}, and 3.85×10^{-3}, respectively.

Comparatively, the Time-VAE exhibited slightly higher reconstruction errors across all the sensor types, indicating a relatively lower precision in capturing the nuances of the time series data. The MST-VAE notably demonstrated considerably higher errors, particularly for the Power and CSS sensors, with MAPE values of 23.4% and 55.1%, respectively.

While the reconstruction error comparisons provide insights into the quality of the reconstructed data, it is essential to note that the primary objective of this study is unsupervised anomaly detection rather than direct model comparison. Hence, further analysis is required to evaluate the effectiveness of each model in detecting anomalies within the company's operational data.

7.2 Anomaly Detection in Sandvik Data

A comprehensive evaluation of the proposed Variational Autoencoder (VAE) for unsupervised anomaly detection in multivariate time series was conducted using dedicated anomaly detection metrics. This assessment aimed to determine the efficacy of the VAE in detecting anomalies within Sandvik's operational data. The following table presents the results of the anomaly detection metrics comparison for the VAE.

Table 7.2 Anomaly detection metrics for the VAE on the labelled Sandvik dataset.

Anomaly Detection				
Model	Sensor	Anomaly detection metrics		
		F1	Recall	Precision
VAE	Power	0.89	0.89	0.89
	Pressure	0.75	0.75	0.75
	CSS	0.92	0.92	0.92

The table demonstrates that the VAE exhibited promising performance in detecting anomalies within the company's operational data. The F1 score, a widely used metric that combines precision and recall, yielded consistently high values for all sensor types. Specifically, the VAE achieved an F1 score of 0.89 for the Power sensor, 0.75 for the Pressure sensor, and 0.92 for the CSS sensor. These results indicate the VAE's ability to identify and classify sensor data anomalies effectively.

Moreover, the recall metric, which quantifies the proportion of true anomalies correctly detected by the model, exhibited excellent performance across all sensor types, with values matching the corresponding F1 scores. This indicates the VAE's capability to capture a significant portion of the anomalies present in the operational data.

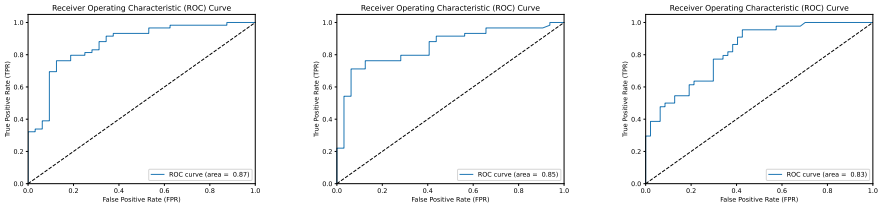
Additionally, the precision metric, representing the accuracy of anomaly detection, mirrored the F1 scores and recall values, further highlighting the robustness of the VAE in minimizing false positive detections.

It is important to note that the anomaly detection evaluation was conducted based on the defined criterion of considering an entire operation as an anomaly. This choice was made for practical reasons, limiting the direct comparison of anomaly detection performance with other models that consider smaller parts of an operation as anomalies.

In summary, the results obtained from the evaluation of anomaly detection metrics confirm the effectiveness of the VAE in identifying anomalies within the multivariate time series data. The high F1 scores, recall values, and precision demonstrate the VAE's ability to reliably detect anomalies, thereby providing valuable insights for monitoring and maintaining the operational integrity of the company's systems.

7.3 AUC-ROC Curve

The proposed anomaly detection mechanism will be analyzed using ROC curves and the area under the curve (AUC). Below is a graph showing the ROC for the three features Power, Pressure and CSS.



(a) Power, AUC = 0.87 **(b)** Pressure, AUC = 0.85 **(c)** CSS, AUC = 0.83

Figure 7.1 The ROC curve for Power, Pressure and CSS

The ROC-AUC results for the three features are illustrated in Figure 7.1. For Power, the AUC is found to be 0.87 (see Figure 7.1a). For pressure, the AUC is slightly lower, at 0.85, as shown in Figure 7.1b. Lastly, for the CSS feature, the AUC stands at 0.83 (see Figure 7.1c). These values indicate that the VAE model performs well in detecting anomalies for all three features, with the best performance for the power feature and the lowest, yet still suitable, for the CSS. A more detailed analysis of anomaly detection can be conducted by examining the true positive rate (TPR), false positive rate (FPR), and thresholds used for classification. The TPR and FPR values for various thresholds reveal the trade-off between sensitivity and specificity that the model must balance for effective anomaly detection.

Power

For the Power, the ROC curve shows a good trade-off between TPR and FPR for the power feature as the threshold increases. The model achieves high TPR values (above 0.81) while maintaining low FPR values (under 0.3). The ROC curve starts from a convex shape and then transitions toward a more straight line as the threshold increases, reflecting consistent detection performance across a wide range of thresholds. This indicates a well-trained model that discriminates between anomalies and well-behaved data points.

Pressure

For the pressure, the ROC curve is predominantly convex. This suggests that while the classifier may not be as efficient in detecting anomalies as the power feature, it still provides a commendable separation between the two classes. However, the FPR could experience a significant increase for higher TPR values. The ROC curve highlights that the model can achieve high TPR values (0.91 or higher) but at the cost of moderately increased FPR values (around 0.56). This might lead to more false positive cases but still signifies adequate detection capability.

CSS

The CSS exhibits slightly different behaviour in its ROC curve. While the model can achieve a TPR value of 1.0, it does so with a significantly higher FPR value of 0.91. This elevated FPR suggests that the model could be more prone to producing false alarms for the CSS feature compared to the other two features. Despite this, the performance of anomaly detection remains satisfactory for practical applications.

7.4 Anomaly score

Before presenting examples of different anomaly scores and their corresponding machine behaviours, it is essential to highlight the underlying concept behind the anomaly score calculation. In this thesis, the anomaly score is derived from the reconstruction's Mean Absolute Percentage Error (MAPE) and is further normalized by the anomaly threshold. Any value above 1 indicates an anomaly, while any value below 1 denotes a non-anomalous instance. However, it is worth exploring whether the anomaly score can provide insights beyond binary classifications, shedding light on the severity of detected anomalies.

A set of example images with varying anomaly scores will be presented to investigate the potential relationship between anomaly scores and the severity of anomalies. These images capture instances where the machine behaviour visibly deteriorates as the anomaly score increases. By visually examining these examples, valuable insights can be gained into the behaviour patterns exhibited by the machine and determine whether there is a correlation between higher anomaly scores and more severe anomalies.

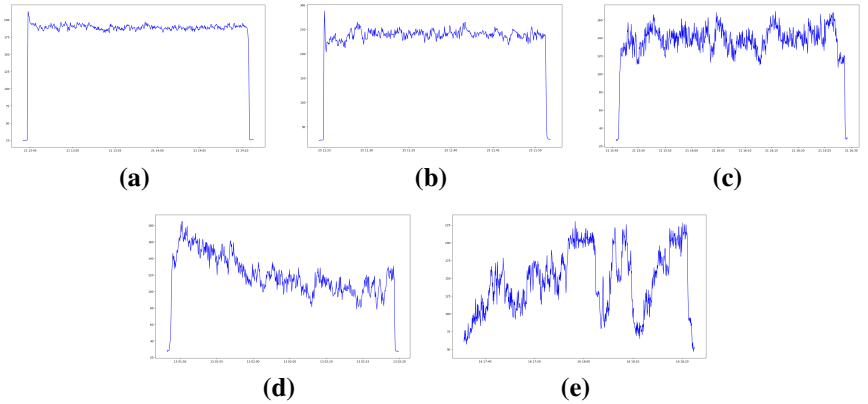


Figure 7.2 **(a)** Power sensor for operation with anomaly score 0.62 **(b)** Power sensor for operation with anomaly score 1.32 **(c)** Power sensor for operation with anomaly score 2.92 **(d)** Power sensor for operation with anomaly score 3.68 **(e)** Power sensor for operation with anomaly score 7.38

The presented example images showcase the observed machine behaviours corresponding to different anomaly scores. As the anomaly score increases, it becomes evident that the machine's performance visibly worsens, characterized by higher variations and increasingly irregular behaviours. These anomalies manifest as deviations from the expected normal operations, indicating a higher severity in the machine's abnormal functioning.

By leveraging these example images, it is reasonable to argue that the anomaly score can provide valuable information beyond a simple binary classification of anomalies. The increasing irregularities and variations observed in machine behaviours as the anomaly score rises a correlation between the anomaly score and the severity of anomalies. Consequently, the anomaly score can indicate the degree of deviation from normal operations, enabling identifying and assessing anomalies based on their severity. When experts at Sandvik were given these examples, they also agreed that the anomaly score examples with higher anomaly scores were exhibiting more strange behaviour.

The fact that higher anomaly scores indicate worse performance is of great practical significance. Since it can point experts and engineers to the most strange behaviour in the sensor data, assessing the severity of anomalies is crucial for effective anomaly management.

7.5 Additional Analysis

In this thesis, the anomaly score introduced is one such metric that could be useful. During the project, sensor data were examined, and technicians were interviewed.

Based on this, trends and patterns have been identified that could be a good starting point for constructing valuable metrics and heuristics. Due to time restraints and a lack of data, the assumptions listed below have yet to be fully verified, but further analysis of these relationships could be a good starting point for establishing future performance metrics. The list of patterns below will be briefly discussed in this section:

- Power and pressure relationship
- Variance
- Deviation from baseline
- Magnitude of deviation in CSS

Power and pressure relationship

The magnitude of Power and pressure are linked. During regular operation, Power and pressure are determined mainly by the cone crusher's resistance when crushing rocks. This leads to a linear correlation between Power and pressure. The constant determining the linear relationship between Power and pressure differs for different crusher models and operating conditions. When the linear relationship no longer holds during crushing, it indicates anomalous behaviour observed by experts at Sandvik.

Variance

High variance is a good indication of faulty behaviour. When crushing rocks with a cone crusher, the machine operates optimally when external conditions are stable. The stone crushers operate best when the feed of rocks is constant, and the machines are sufficiently fed with rocks. When the feed rate and the type of material fed to the machine vary, the power and pressure signal increase variance. No specific threshold values have been identified, as the importance of variance may be relative to the context and specific machine.

Deviation from baseline

During this project, the anomaly detection algorithm operated without taking the model of the machine into account and the difference in scale was mitigated using normalization. This was due to limited data and many different cone crusher models. When experts analyzed the sensor data, they were observed to compare the Power and pressure magnitude to the baseline for the cone crusher model being analyzed. Each machine has a designated baseline, determined during the design of the cone crusher model, that it is intended to operate around. Cone crushers with power or pressure readings either far above or far below the baseline usually displayed indications of faulty operation.

Magnitude of deviation in CSS

The CSS changed slowly during normal operations as the cone crusher mantle was incrementally worn down. When the CSS varied much more than one millimetre quickly, it indicated that the gap where the stones exited the crusher was enlarged by causes other than wear and tear. One example is if a large and uncrushable object enters the stone crusher, it can forcefully activate the hydraulic dump valve when the mantle comes in contact with the uncrushable object and cannot compress it. Another example is when the operator intentionally makes the gap larger for some reason, such as dislodging an uncrushable object stuck in the machine or changing to the size of the produced stones.

8

Conclusions And Future Work

8.1 Improvements, Limitations and Future Work

There are many ways to improve the project. For instance, improvements can surely be made with more data and more elaborate performance metrics.

In this work, anomalies were defined at the level of entire operations rather than smaller temporal segments. This choice may limit the ability to directly compare the anomaly detection performance with models considering more fine-grained anomalies. Future research should explore alternative approaches to defining and detecting anomalies at different levels of granularity, allowing for more detailed analysis and comparison with other models. A limited amount of labelled anomalies guided this choice, but with a large data set of labelled anomalies, semi-supervised or supervised approaches could be applied. Anomaly detection on smaller temporal segments would allow for easier comparisons to state-of-the-art anomaly detection algorithms to assess the strengths and limitations of the proposed VAE-based approach and further validate its effectiveness.

More elaborate performance metrics would make incorporating domain knowledge into anomaly detection methods easier. It would be worthwhile to explore ways to incorporate domain knowledge and expert insights into the anomaly detection process, leveraging domain-specific information to enhance the accuracy and robustness of the model. In constructing the model, normalization was used to account for the different magnitudes of power and pressure in different cone crushers. If individual baselines for power and pressure were incorporated for each crusher type, it would improve the model's performance.

Investigate more advanced neural network architectures, such as Transformer-based models or Graph Neural Networks, to capture complex temporal and contextual dependencies in the multivariate time series data. In recent years transformer-based architectures have established themselves and produced excellent results in many domains. One of the limits of transformer-based architectures is the sheer

amount of data needed to train them effectively. With the data I had access to, this type of architecture would not be feasible. However, since Sandvik cone crushers are worldwide, the data could be aggregated to make this approach viable.

When analyzing models, the more metrics used, the better. Additional metrics in the performance analysis, like the RP-AUC and the reverse percentile distance, would also have been a helpful way to analyse the model's performance further. These metrics were not a part of the thesis due to time constraints.

Skewed Anomaly Dataset

Sandvik collected a set of ASRI files for use in the project. Though they contained more operation data, only a week of operations was taken from each ASRI file. A subset of the ASRI files was marked carefully by domain experts as well-behaved machines. Due to the nature of the task, the VAE was trained exclusively on well-behaving data from these ASRI files. Since these ASRI files were used in the training of the VAE, they were not used for anomaly detection.

Training the anomaly detection model on the same data set used for training and validation would have resulted in overfitting. Overfitting occurs when a model becomes too specialized in capturing patterns from the training data, making it less capable of generalizing to new and unseen data. In anomaly detection, overfitting would limit the model's ability to detect anomalies in real-world scenarios effectively.

During the project, experts at Sandvik labelled a set of operations anomalous. This labelling process involved introducing the operations to the experts who assessed their anomalousness based on their domain knowledge. One data set was used to set the anomaly threshold by selecting the threshold that achieved the best F1 score, a measure that balances precision and recall. This threshold was validated by testing its performance on a separate data set.

It is important to note that the anomaly detection dataset used for setting the threshold and validating the anomaly detection algorithm exhibited an abundance of anomalous operations. This inflation of anomalies was a direct consequence of excluding the well-behaved data that was used exclusively for the training of the VAE. Therefore, the resulting dataset does not necessarily reflect the actual frequency of anomalous behaviour. When there is an inflation in the number of anomalies in the dataset, it can negatively impact the accuracy of the F1 score as a metric for evaluating the anomaly detection algorithm. In the case of an inflated number of anomalies, the algorithm may be biased towards classifying more instances as anomalies, resulting in a higher recall. This is because the algorithm is more likely to label instances as anomalies, given the abundance of anomalies in the dataset. However, this increased recall may come at the expense of precision, as the algorithm may produce more false positives. However, since the anomaly detection had high recall and precision and could generalize for anomaly scores above one, the model is tracking Sandvik's notion of an anomaly. To mitigate these limitations, future work

should focus on collecting a larger-scale dataset with a more balanced representation of well-behaved instances and anomalous behaviours. This would enable the VAE to be trained in a broader range of well-behaving data while ensuring sufficient data to accurately represent the frequency of anomalous behaviour.

8.2 Conclusions

In conclusion, this thesis encountered challenges stemming from the development of performance metrics and limitations in the available data. However, these challenges were addressed by applying unsupervised learning techniques and adopting a solvable anomaly definition, ultimately providing value to Sandvik. The inclusion of a state algorithm in the preprocessing stage significantly contributed to the success of the anomaly detection process by narrowing the focus to relevant intervals in the data. It is important to note that the limited data availability and the constraint of training the VAE solely on well-behaved data resulted in a skewed dataset with a disproportionate number of anomalies. This imbalance may have impacted the results' reliability and generalizability, particularly regarding the F1 score, recall, and precision metrics. However, the robust performance of the method in terms of the ROC-AUC metric lends greater credibility to the obtained results.

The ROC-AUC metric's ability to handle imbalanced class distributions is particularly advantageous for evaluating models trained on datasets where anomalies are significantly rarer than normal. The ROC-AUC metric comprehensively evaluates the model's capacity to distinguish between the two classes, regardless of their imbalance, by considering various classification thresholds. Consequently, the trustworthiness of the results is reinforced. Furthermore, the observation that the generated anomaly scores correlate with the severity of the anomalies suggests that the anomaly detection approach extends beyond binary classification. This indicates that the method has the potential to provide engineers with valuable insights into anomalous instances within the sensor data. Moreover, even for machines exhibiting consistently poor performance, the method remains effective in identifying the most severe anomalies specific to each circumstance.

In summary, this method demonstrates its ability to identify and notify engineers of anomalous instances in sensor data, showcasing its practical utility for anomaly detection in industrial settings. Integrating the ROC-AUC metric and the correlation between anomaly scores and severity substantiate the method's reliability and potential to extend beyond binary classification. As a result, this work presents a good foundation for future advancements in unsupervised anomaly detection in multivariate time series data for industrial applications.

Bibliography

Aggarwal, C. C. and C. C. Aggarwal (2017). *An introduction to outlier analysis*. Springer.

Alla, S., S. K. Adari, S. Alla, and S. K. Adari (2019). "Introduction to deep learning". *Beginning Anomaly Detection Using Python-Based Deep Learning: With Keras and PyTorch*, pp. 73–122.

An, J. and S. Cho (2015). "Variational autoencoder based anomaly detection using reconstruction probability". *Special lecture on IE* **2**:1, pp. 1–18.

Andrieu, C., N. De Freitas, A. Doucet, and M. I. Jordan (2003). "An introduction to mcmc for machine learning". *Machine learning* **50**, pp. 5–43.

Baur, C., S. Denner, B. Wiestler, N. Navab, and S. Albarqouni (2021). "Autoencoders for unsupervised anomaly segmentation in brain mr images: a comparative study". *Medical Image Analysis* **69**, p. 101952.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Burgess, C. P., I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner (2018). "Understanding disentangling in β-vae". *arXiv preprint arXiv:1804.03599*.

Chatterjee, S., A. Sciarra, M. Dünnwald, P. Tummala, S. K. Agrawal, A. Jauhari, A. Kalra, S. Oeltze-Jafra, O. Speck, and A. Nürnberger (2022). "Strega: unsupervised anomaly detection in brain mris using a compact context-encoding variational autoencoder". *Computers in Biology and Medicine* **149**, p. 106093.

Chui, M., B. Hall, H. Mayhew, A. Singla, and A. Sukharevsky (2022). *The state of ai in 2022—and a half decade in review*. `https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2022-and-a-half-decade-in-review`.

Davenport, T. and J. Fitts (2021). "Ai can help companies tap new sources of data for analytics". *Harvard Business Review*. URL: `https://hbr.org/2021/03/ai-can-help-companies-tap-new-sources-of-data-for-analytics`.

De Myttenaere, A., B. Golden, B. Le Grand, and F. Rossi (2016). "Mean absolute percentage error for regression models". *Neurocomputing* **192**, pp. 38–48.

Desai, A., C. Freeman, Z. Wang, and I. Beaver (2021). "Timevae: a variational auto-encoder for multivariate time series generation". DOI: 10.48550/ARXIV.2111.08095. URL: https://arxiv.org/abs/2111.08095.

Drai, D. (2021). *Why ai-driven analytics is essential for data-driven decision-making*. https://www.forbes.com/sites/forbestechcouncil/2021/12/27/why-ai-driven-analytics-is-essential-for-data-driven-decision-making/.

Dumoulin, V. and F. Visin (2016). "A guide to convolution arithmetic for deep learning". *arXiv preprint arXiv:1603.07285*.

Fährmann, D., N. Damer, F. Kirchbuchner, and A. Kuijper (2022). "Lightweight long short-term memory variational auto-encoder for multivariate time series anomaly detection in industrial control systems". *Sensors* **22**:8, p. 2886.

Freund, Y. and D. Haussler (1991). "Unsupervised learning of distributions on binary vectors using two layer networks". *Advances in neural information processing systems* **4**.

Ganguly, A. and S. W. Earp (2021). "An introduction to variational inference". *arXiv preprint arXiv:2108.13083*.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep learning*. MIT press.

Higgins, I., L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner (2017). "Beta-vae: learning basic visual concepts with a constrained variational framework". In: *International conference on learning representations*.

Kingma, D. and J. Ba (2014). "Adam: a method for stochastic optimization". *International Conference on Learning Representations*.

Kingma, D. P. and M. Welling (2013). "Auto-encoding variational bayes". *arXiv preprint arXiv:1312.6114*.

Kiranyaz, S., O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman (2021). "1d convolutional neural networks and applications: a survey". *Mechanical systems and signal processing* **151**, p. 107398.

Oreshkin, B. N., D. Carpov, N. Chapados, and Y. Bengio (2019). "N-beats: neural basis expansion analysis for interpretable time series forecasting". *arXiv preprint arXiv:1905.10437*.

Park, D., Y. Hoshi, and C. C. Kemp (2018). "A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder". *IEEE Robotics and Automation Letters* **3**:3, pp. 1544–1551.

Pham, T.-A., J.-H. Lee, and C.-S. Park (2022). "Mst-vae: multi-scale temporal variational autoencoder for anomaly detection in multivariate time series". *Applied Sciences* **12**:19. ISSN: 2076-3417. DOI: 10.3390/app121910078.

Rijsbergen C.J., van (1995). *Information Retrieval*. draft.

Russell, S. J. (2010). *Artificial intelligence a modern approach*. Pearson Education, Inc.

Sandvik (n.d.). *Cone crusher basics in 4 minutes*. https://www.rockprocessing.sandvik/en/crushology/the-knowledge-hub/crushing-chambers/cone-crusher-basics-in-4-minutes/.

Schemmer, M., J. Holstein, N. Bauer, N. Kühl, and G. Satzger (2023). "Towards meaningful anomaly detection: the effect of counterfactual explanations on the investigation of anomalies in multivariate time series". *arXiv preprint arXiv:2302.03302*.

Su, Y., Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei (2019). "Robust Anomaly Detection for Multivariate Time Series through Stochastic Recurrent Neural Network". In: *Proceedings of the 25th International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 2828–2837. ISBN: 978-1-4503-6201-6. DOI: 10.1145/3292500.3330672.

Ulger, F., S. E. Yuksel, and A. Yilmaz (2021). "Anomaly detection for solder joints using β-vae". *IEEE Transactions on Components, Packaging and Manufacturing Technology* **11**:12, pp. 2214–2221.

Zeiler, M. D., D. Krishnan, G. W. Taylor, and R. Fergus (2010). "Deconvolutional networks". In: *2010 IEEE Computer Society Conference on computer vision and pattern recognition*. IEEE, pp. 2528–2535.

Zhou, L., W. Deng, and X. Wu (2020). "Unsupervised anomaly localization using vae and beta-vae". *arXiv preprint arXiv:2005.10686*.

A

1D Convolution Visualization

Following is a visualization of 1D convolution with stride 2, input size 5 and kernel size 3:

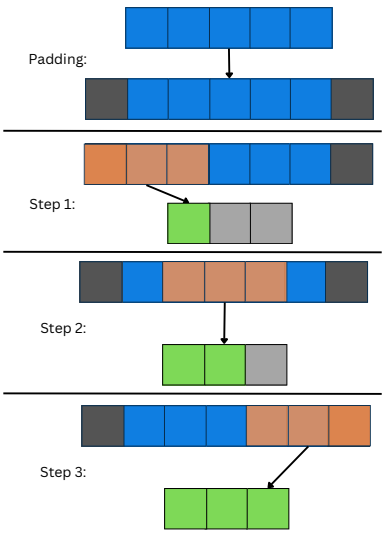


Figure A.1 Visualization of 1D convolution. Blue represents the input array, the dark grey represents 0. The orange is the sliding filter, and light grey is the not yet assigned element in the output. The green is the assigned elements in the output

B

1D Transposed Convolution Visualization

Following is a visualization of 1D transposed convolution with stride 2, input size 2 and kernel size 3;

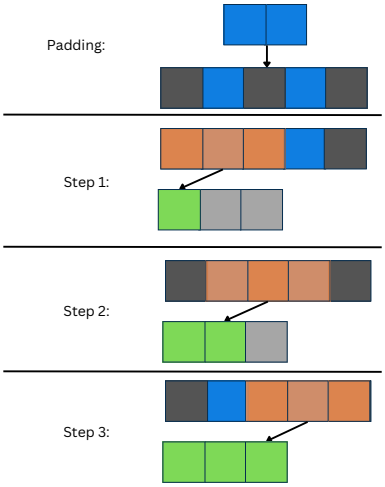


Figure B.1 Visualization of 1D convolution. Blue represents the input array, the dark grey represents 0. The orange is the sliding filter, and light grey is the not yet assigned element in the output. The green is the assigned elements in the output

C

State Algorithm Visualization

Below is an example of how the state algorithm segments the sensor data.

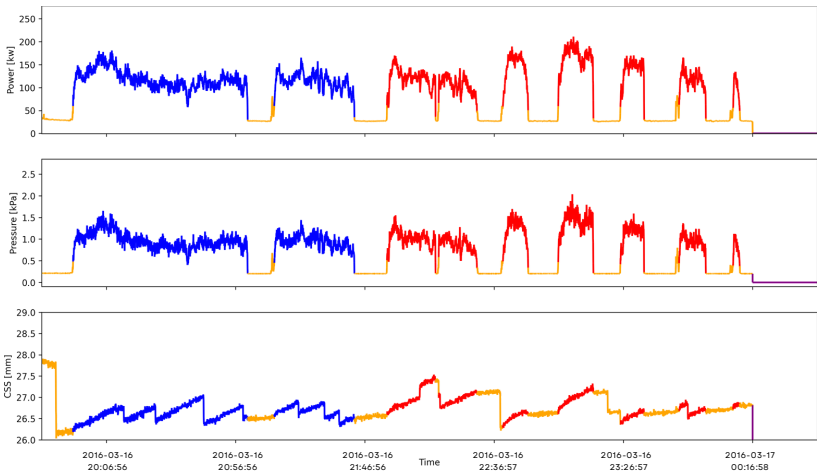


Figure C.1 Sensor data segmented by the state algorithm. The blue segments are the continuous operations, and the red segments are the discontinuous operations. The yellow segments are the idle segments, and the purple segments are the off segments.