Unsupervised Deep Anomaly Detection in a Recirculating Aquaculture System

by

William Robinson

A Thesis submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, MB

Abstract

An unsupervised deep anomaly detection system is implemented to augment the water quality monitoring system used at a recirculating aquaculture system (RAS) facility. Its purpose is to increase the system's anomaly detection capabilities by improving its accuracy and decreasing the timeframe in which anomalies can be detected. Quick and precise detection of abnormalities leads to earlier action to reduce mortalities within the fish population, or prevent them altogether.

The machine learning model introduced in this work, given the name aM-SCRED or *adaptive Multi-Scale Convolutional Recurrent Encoder-Decoder*, is an expansion of the MSCRED model featured in previous work by Zhang et al.[1] This model is a spatio-temporal network (STN) composed of stacked CNNs and RNNs, structured in an autoencoder architecture. This configuration is capable of learning what characterizes normal behaviour within a multivariate timeseries dataset, which can thereafter be leveraged to detect abnormal behaviour, which may indicate a problem in the system.

Using data obtained from the monitoring system at the RAS facility, aM-SCRED is able to outperform its predecessor in terms of anomaly detection performance (measured in terms of Recall, Precision and F1 score). Recall scores of up to 97% were achieved, as well as F1 scores of up to 94%. It also outperforms its predecessor in root cause identification (RCI), achieving accurate prediction rates of $\sim 70\%$, compared to $\sim 50\%$ using the model from Zhang et al.[1] The improved results are made possible due to modifications

which enable the model to adaptively select, on a per-dataset basis, different signature matrix generating strategies, model structure parameters, anomaly scoring methodologies, and root cause scoring methodologies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Fish production represents a significant portion of the world's protein production, reaching 17% of global production in 2014. This portion has been growing ever since. Over the last couple of decades, aquaculture-based production has steadily outpaced capture-based operations and currently represents nearly half of total fish protein production. This ratio is continually growing as well. Projections by the Food and Agriculture Organization (FAO) of the United Nations indicate that by 2030, aquaculture production will represent the majority of fish protein production, reaching an estimated 109 million tonnes annually. This represents an increase of 37% from 2016 levels, to 68% of total fish protein production by 2030. [2]

The aquaculture industry is a growing industry that faces unique challenges that are non-existent in capture-based operations. Namely, smaller margins on products sold due to extra costs associated with meeting feeding

requirements and increased mortality rates which result in greater losses due to stock depletion. Some mortalities are due to failings in the biology and genetics of individual specimens, which are more difficult to predict and prevent. However, mortalities related to poor living conditions are preventable, if these detrimental conditions are detected ahead of time and appropriately addressed. These shortcomings, which are especially true of in-land aquaculture operations, risk hurting the scalability of the industry if not addressed and rectified in the near-future.

Advances in new technologies, such as machine learning (e.g. sensors, monitoring systems, anomaly detection, and automation) and process optimizations (e.g. defining standard operating procedures to respond to abnormalities in the system) can be adapted to and implemented in aquaculture systems in order to reduce mortality rates. Reducing such unnecessary losses will permit this industry to scale in a healthy way and allow it to meet increasing global demand of fish protein. This work presents a novel implementation of an unsupervised anomaly detection system that works with existing sensors and monitoring systems, in order to detect abnormalities in the RAS, in an effort to reduce the system's mortality rate.

In the remainder of this chapter, the motivation behind this work is described in Section 1.1, followed by an overview of the research conducted, in Section 1.2. Some context on timeseries anomaly detection is presented in Section 1.3. Finally, the structure of the remainder of this thesis is laid out in Section 1.4.

1.1 Motivation

The research in this work focuses on the implementation of an unsupervised deep anomaly detection (DAD) algorithm capable of anomaly detection in multivariate timeseries data obtained from sensors installed in a modern high-tech aquaculture facility. Unsupervised machine learning in this context is key, because it provides the ability for the algorithm to learn to recognize a normal system state without requiring labelled training data. It is also able to adjust its understanding of a normal system state as the system evolves in time (e.g. evolution of system conditions as cohorts of fish grow and undergo physiological changes).

The purpose of the implementation of this machine learning algorithm is to continuously compare the current system state to the learned normal state and detect abnormal conditions in the system as they arise (i.e. detect anomalies). In the event of an anomaly being detected, system operators are alerted in order to elicit the appropriate response. With enough notice, actions taken by the system operators can prevent unnecessary fish deaths.

Furthermore, snapshots of the system state can be recorded in order to characterize the system at any point in time. This capability falls in line with food traceability applications which track facility health over a given period of time. This level of traceability in food production is increasing in demand across the food production industry and is also on track to become a regulatory requirements in the future. Therefore, having this capability

built-in ahead of time is a bonus.

The algorithm implemented in this work is based on the latest advances in spatio-temporal networks (STN), a subtype of deep anomaly detection commonly used on multivariate timeseries data. The architecture in question is based on the MSCRED model, found in previous work by Zhang et al., but this implementation adds novel modifications on certain components in order to increase its anomaly detection performance in the aquaculture context. [1, 3]

1.2 Overview of the Research

The algorithm implemented in this research is based upon the *Multi-Scale Recurrent Encoder Decoder* model. MSCRED utilizes a stack of different neural network components, configured in an autoencoder architecture. This mixture of components can characterize both the spatial dependency between multiple timeseries (using Convolutional Neural Networks, or CNNs), as well as the temporal dependency (using Recurrent Neural Networks, or RNNs, and Attention). [1]

The MSCRED model was chosen as the basis on which to build the implementation in this work, as it has shown its ability in previous work to outperform other unsupervised techniques in anomaly detection of timeseries data.[1] This performance is measured using the standard metrics of Precision (Pre), Recall (Rec), and F1 score. Furthermore, it has also demonstrated

competence in root cause identification (i.e. identifying which individual timeseries are responsible for a given anomaly).[1]

The primary objective of this research is to implement and optimize the structure of MSCRED to perform anomaly detection in the context of a recirculating aquaculture system (RAS). In doing so, some novel modifications are made to certain components of the model in order to increase its adaptability to datasets, in the hope that the model might adapt itself to different aquaculture systems in the future, requiring only trivial modifications to the implementation itself. These modifications are detailed in Chapter 3. As such, this modified model is given the name *adaptive Multi-Scale Recurrent Encoder Decoder*, or aMSCRED. Various model hyperparameters of the implementation of Zhang et al. were empirically determined for their specific dataset. These hyperparameters are identified and tuned to find the optimal values for the RAS dataset studied in this research. They are left as dynamically tunable parameters to allow future expansion into other applications.

The secondary aim of the aMSCRED implementation is to outperform its predecessor in terms of anomaly detection performance (measured via Precision, Recall, and F1 score) with respect to RAS timeseries data. This comparison is achieved by matching the hyperparameters presented in the MSCRED paper to do a head-to-head comparison of results.[1]

A third research objective is to expand the root cause identification strategy elaborated on in [1]. This research proposes two additional metrics to identify root causes and compares their performance to the metric presented

in [1].

Another question of interest relates to the preprocessing of the model's input data. The input data used to train the model is not the raw time-series data itself, rather it is derived data (called *signature matrices*) which characterize pairwise correlations between each timeseries pair over multiple windows of fixed length. This research proposes alternate correlation functions used when generating the signature matrices, namely the Pearson correlation coefficient and time-weighted correlation functions (i.e. time steps further in the past have lesser effect on the correlation measure in the signature matrix). They, as well, are compared with the method presented in the implementation by Zhang et al. [1]

1.2.1 Further Context

When implementing anomaly detection algorithms, there exists an uncircumventable trade-off between their ability to minimize false positives (FP) and false negatives (FN). In the context of aquaculture, false negatives have the potential to exert vastly more negative impact on production since unchecked anomalies can lead to increased mortality rates. As such, when implementing and training the model for the context of the RAS to address the research objectives above, results that minimize false negatives at the expense of an increase in false positives are favoured. This rationale will be elaborated on further in a later section.

1.3 Timeseries Anomaly Detection

What are anomalies? In the general sense, they are abnormalities (or outliers) present within a given dataset; they are data points that noticeably do not align with the majority of other datapoints present in the dataset.

In the context of timeseries data, the dataset consists of sequential data that varies in time. Anomalies in this context consist of data points that break a repeating sequence or range of expected values, based on the remaining datapoints in the dataset. An example of this is shown in Figure 1.1.

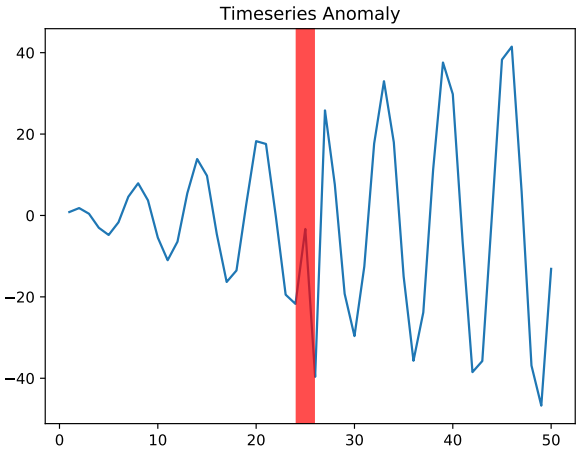

Figure 1.1: Anomaly in a timeseries. Anomalous timesteps (highlighted in red) do not follow the greater general trends of the remainder of the timeseries.

1.3.1 Deep Anomaly Detection (DAD)

Deep-learning is a subset of machine learning techniques that has grown in popularity in recent years, outpacing traditional machine learning techniques in many applications, as the scale of data input into the system increases.[4]

The popularity of deep-learning based anomaly detection techniques is no exception. With the prevalence of timeseries data increasing at a staggering rate, DAD techniques have become increasingly widespread in the past decade. DAD techniques have been applied to intrusion detection, fraud detection (e.g. banking, insurance, healthcare), malware detection, medical anomaly detection, industrial anomaly detection, and timeseries anomaly detection, to name a few.[4]

The challenge when dealing with multivariate timeseries is that both the spatial and temporal characteristics of the system must be examined; that is, both the pairwise correlation of each sensor in a given segment of time and the evolution of each single timeseries through time must be well understood.[1]

The type of DAD implemented in this research is unsupervised multivariate timeseries anomaly detection. Unsupervised because of its ability to learn the inherent structure of what characterizes data as normal versus anomalous and doing so in a cost-effective manner. It does not require annotated data, which is a time consuming and expensive task to undertake.[4]

1.4 Thesis Structure/Organization

The remainder of this thesis is structured as follows. Chapter 2 presents background information on aquaculture, its economic importance, as well as physical parameters of a RAS and fish physiology that dictate the health of the fish populations contained in the RAS. Chapter 3 presents machine learning background, including theoretical concepts utilized in aMSCRED, followed by a detailed breakdown of the components of aMSCRED, contrasting the modifications made in the context of this research with respect to the MSCRED implementation of Zhang et al.[1] Chapter 4 describes the particular recirculating aquaculture system (RAS) and sensor system from which the dataset used in this research was obtained, as well as the experimental methods undertaken. These include a characterization of the datasets used, the data preprocessing they are subject to, the model hyperparameter tuning process, the training validation and testing process, the anomaly scoring and detection methodology (i.e. when to predict a given input as anomalous), the root cause identification methodology, as well as defining the metrics used to evaluate anomaly detection performance. Chapter 5 summarizes results obtained for each of the evaluation metrics as well as overall anomaly detection on the RAS dataset. Chapter 6 concludes this work and evaluates to what degree each research objective was attained (or not).

The Appendixes contain detailed information that was not essential to the understanding of the research presented in this work, but serve as additional

context to interested parties. Appendix A details how the monitoring system used at the research facility functions and how it interacts with the staff, as well as how aMSCRED fits into it. Appendix B looks at considerations for implementing aMSCRED in a production system (i.e. using it as an online anomaly detection system). Appendix C details the implementation of the aMSCRED model in Tensorflow, presenting details, diagrams, and specifications of the model in code.

Chapter 2

Aquaculture Background

2.1 Relevance of the Aquaculture Industry

This section discusses the importance of the in-land aquaculture industry on a global scale, as well as challenges it faces, establishing the importance of developing technology for the aquaculture industry. This is to justify committing energy and resources to push along research and innovation in this domain.

2.1.1 Global Production and Consumption

Global fish production peaked at roughly 171 million tonnes in 2016. Aquaculture represented 47% of production that year, up from 25.7% in 2000, achieving an average growth rate of 5.8% annually.[2]

Since 1961, the average annual increase of fish consumption (3.2%) has

been twice that of human population growth (1.6%). Demand for consumable fish products is exponential growth of the human population, requiring significant scale-up in fish production capacity, world-wide, in order to meet growing market demand.[2]

However, while the global demand for fish products grow, capture-based fisheries production have stagnated since the 1980s, as seen in Figure 2.1.

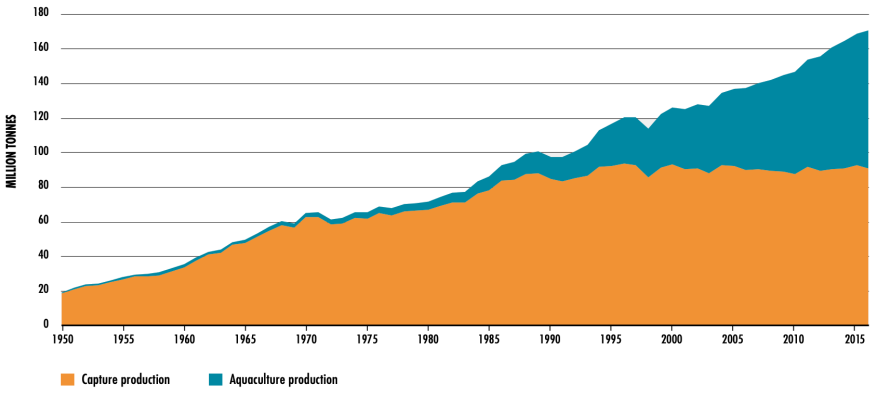

Figure 2.1: World fish production. In orange, capture-based portion and in blue, aquaculture-based portion of production. Figure obtained from [2]

As time passes, in-land aquaculture production is projected to represent an increasing share of production capacity and will become the primary source to meet the increased global demand for aquatic protein.[2]

2.1.2 Advantages and Risks of Aquaculture

In 2016, roughly 60 million people are estimated to be engaged in fish production globally, of which 20 million in aquaculture-based operations and 40 million in capture-based operations.[2] Both market segments represent

a similar production capacity, yet aquaculture achieves it with only half the workforce. This indicates an inherent efficiency for aquaculture-based operations in terms of labour requirement. This fares well with respect to the expected scale-up of the industry over the next decade.

This efficiency is further increased with the ability to introduce high-tech solutions (such as facility automation, sensing and monitoring systems, etc...) which further extend the capacities and capabilities of existing facilities and their workers. Research and advances in this technology holds the potential for aquaculture-based operations to further increase is efficiencies, solidifying its position to become the primary source of production of aquatic protein.

However, aquaculture comes with its set of risks that either do not exist or are not as prevalent in capture-based operations. Namely, water conditions must be closely monitored and maintained within a given operating range to ensure good fish health. If a problem arises with any equipment (pumps, filters, etc...) the living conditions of the system can rapidly deteriorate and lead to increased mortality rates. This risk can be mitigated, using electronic sensors and monitoring systems which are becoming increasingly common-place across all industries.

2.1.3 Existing Monitoring Solutions

Currently, existing systems available on the market can monitor multiple system parameters (e.g. temperature, dissolved oxygen, pH), set thresholds within which these parameters should remain, and alert operators (via

some computer interface or alarms) or initiate automated tasks (e.g. turn on a pump) whenever these parameters leave their normal bounds. The Pentair line of monitoring products, IPI Singapore Fish Farm Monitoring System, MonitorFish, and RTAqua, all consist of a payload of different sensors which tie in with some software suite, visualization tools, integration with programmable logic controllers (PLC) (Pentair and IPI), but utilize the traditional threshold-only monitoring approach.

The drawback of this type of monitoring system is that it is not capable of detecting the complex combination of issues which may be happening across multiple system parameters. Looked at on an individual basis, parameters may look normal (i.e. everything is within normal ranges) but may in fact be dangerous when combined.

An arbitrary example of such an event might be a dead fish that is caught in a pump line, reducing system flow and simultaneously leaching undesirable chemicals into the water. This might not be noticeable in readings on any one individual sensor and could lead to nitrogen poisoning of other fish in the system, if gone unnoticed.

Another such arbitrary example is that a sensor begins to malfunction and returns incorrect readings that appear normal to the monitoring system when in fact they are not.

Sources of systematic errors are also a danger to such monitoring systems. For example, biofilms accumulate on sensor probes over time and bias sensor readings. The offset readings could appear to still be within normal ranges

when in fact they are not.

The scenarios presented above are far too complex to program into a traditional rules-based and threshold-based monitoring systems as they are multivariate issues that arise in unpredictable and unforeseeable manner, manifesting symptoms across multiple sensors. Therefore, it is desirable and cost effective to develop an unsupervised deep anomaly detection system that is constantly learning what a normal system state looks like and using that knowledge to detect abnormalities, ideally before they become noticeable to traditional monitoring systems (or even the trained-eye of employees) and elicit appropriate interventions in order to reduce fish mortalities, or prevent them altogether.

2.2 Species Selection

An important decision when designing an aquaculture system is the species of fish it is designed for, since every species has its own set of needs and requirements. Stocking density is an important requirement to consider, as it determines the total volumetric capacity of the system by setting an upper limit of how many kg of fish can be produced per m^3 of tank volume.

An equally critical consideration is the market price of the species per kg versus its overall cost of production per kg of fish mass, as this will dictate the margins, which is the make-or-break factor for any potential aquaculture operation. The principal factor affecting the margin is the feed conversion

ratio (FCR) of the species, since feed represents the single largest variable cost in aquaculture. FCR is defined as the mass of feed necessary to add a unit of mass onto the fish; for example, an FCR of 1.3:1 means that for every 1.3kg of feed a fish will consume, it will gain 1kg of mass.[5]

2.2.1 Arctic Char

The species of fish at the facility where this research is conducted is Arctic Char. Arctic Char is an appropriate candidate for in-land aquaculture, as it currently has limited supply and is perceived as a high-value species in the market, yielding high demand and high wholesale price. Furthermore, it is an efficient species to farm, with a FCR approaching 1:1 in ideal conditions. The broad body shape and small head give fillet yield around $7-8\%$ higher than rainbow trout.[5] Another favorable characteristic of Arctic Char is its comfort in higher than normal stocking densities, with some facilities achieving up to 120kg/m^3. This means less system volume is required to produce the same output weight compared to other species (e.g. salmon and trout).[5] They also have a high survivability in short-term exposure to low dissolved oxygen concentrations. Also, being an arctic species, they are also cold-water resistant, maintaining a maximal growth rate in the range of 10°C to 15°C.[5]

Although this species is a great candidate for in-land aquaculture, it comes with challenges. For example, eggs and fry are expensive due to low hatchery supply and broodstock development is in its infancy in North Amer-

ica. They also do not feed as aggressively as other species, (e.g. rainbow trout). Therefore, feed must be delivered gradually and slowly throughout the day in multiple passes. It is not a very well-known species in the global market as of yet either, limiting retail demand to northern geographies, where it is most commonly known. [5]

2.3 Fish Physiology

The health of fish depends on the water in which they live. Many factors herein dictate quality of life, most notably water temperature and dissolved oxygen (DO). Other important factors are pH, oxygen reduction potential (ORP) and concentration of fish waste, found as dissolved ammonia and suspended solids (TAN and TSS respectively).

Every species of fish has different levels of these factors that they can endure and surpassing these levels for prolonged durations leads to increased erratic behaviour, disease, and mortality rates.

Many of these factors are correlated due to physical and chemical interactions occurring between them in the water, making maintaining a balance a complicated task. Small changes in one factor can initiate a cascade of larger changes in other factors. For example, maximal DO concentration is limited by water temperature. Thus, as water in the system becomes warmer, it also becomes inherently less oxygenated. Furthermore, increased water temperature increases the metabolic rate of the fish, increasing their oxygen

consumption, further decreasing dissolved oxygen supply. Thus, increasing the water temperature beyond a certain temperature can lead to runaway hypoxia in the fish population.

For this reason, aquaculture facilities must monitor and maintain appropriate values of these system parameters to ensure the efficiency of their production is maintained. Failure to do so leads to increased mortalities. These parameters, as well as interactions between them, are explained in more detail in the remainder of this section.

2.3.1 Dissolved Oxygen

Like most animal species on Earth, fish require oxygen intake to survive. The primary site at which oxygen uptake occurs is at the gill membrane. The efficiency of the gas exchange at this site depends on oxygen tension gradient (partial pressure) between the surrounding water and the fish's blood. The oxygen tension depends on concentration of DO (Dissolved Oxygen), the water temperature, and atmospheric pressure.[6] For salmonoids, such as Arctic Char, acute limit DO is anything less than 2mg/L (i.e. lethal level).[6, 7]

Levels of oxygen delivery below the metabolic requirements are detrimental to the fish, leading to hypoxia. Insufficient oxygen delivery, especially to the brain, for an extended amount of time leads to death.[6] Exposure to prolonged periods of hypoxic conditions have also been shown to increase occurrence of disease, a reduction in growth rate, and interruption and alter-

ation of the reproductive cycle. Evidently, such stresses should be avoided in a commercial setting due to financial losses.

As mentioned earlier, DO levels in the water are primarily limited by the temperature of the water. Solubility of oxygen in water decreases as the temperature is increased. Therefore, the water temperature will impose an inherent limit on dissolved oxygen concentration.

2.3.2 Water Temperature

Most fish are cold-blooded species, that is, their metabolic rate is dictated by their body temperature. Increasing the water temperature linearly increases the metabolism of the fish, requiring more oxygen and more feed to survive. This also diminishes its FCR. This translates to a loss in efficiency in protein-to-protein conversion and increases feed costs (i.e. diminishing margins).[8]

Arctic Char in particular can only survive in a range of 0°C to 16°C, but are able to tolerate short bursts of temperature near their thermal limit, 20°C. Prolonged exposure to this thermal limit temperature has negative impacts on both the immune response (making it prone to disease) and elicit stress response (increasing risk of premature mortality or aggressive behaviour). In turn, this exposure yields increased mortality rates across subjected populations.

2.3.3 TDS, TSS, and Ammonia

Fish waste is produced in the form of ammonia (NH_3) and solid waste. Both forms of waste are detrimental to fish health, however ammonia poisoning is the major concern. Both their concentrations need to be kept below a threshold via filtration units.

A certain portion of dissolved ammonia becomes ammonium (NH_4^+), depending mostly on the pH of the water. Ammonia is more toxic to fish than ammonium. In general, ammonia is more prevalent at higher pH values. Care must be taken to keep total ammonia levels low in the system and to swing the balance in favour of ammonium until it can be filtered out. This is typically achieved by maintaining pH near the lower end of the healthy threshold (more acidic).

The solid waste goes two routes once excreted. A portion of the waste is dissolved, Total Dissolved Solids (TDS), and are removed using biofilters. The remaining solids, Total Suspended Solids (TSS), remain in suspension and are removed from the system using a combination of filters, skimmers, and settling tanks.

2.3.4 pH and ORP

The hydrogen potential, commonly referred to as pH, is the measure of activity level of hydrogen ion exchange in a water-based solution. It is measured on a scale of 0 to 14, where 0 is more acidic and 14 is more alkaline.

Pure water at room temperature is pH neutral, with a pH of 7. Typically, fish can survive anywhere in the range of 5 to 8, depending on species preference, however, it is preferable to stay near the lower-end of this range to maintain an optimal ammonia/ammonium balance.

Oxidation Reduction Potential (ORP), also called Redox, is the activity of oxidizers and reducers in a solution relative to their concentration. Oxidizers accept electrons, and reducers lose electrons. It is measured as an electric potential, in millivolts (mV). Examples of oxidizers that are commonly found in aquaculture are chlorine and hydrogen peroxide. Among other things, ORP is used as an indicator for biological health of the water; lower ORP values promote growth of undesirable biological agents such as algae (below 200 mV), while maintaining higher values (between 250 mV and 400 mV) will prevent such growth and lead to improved fish health. An upper limit for aquaculture is 700 mV as anything above this level will sterilize all life forms in matter of minutes (fish included). Factors linked to low ORP value are low DO levels, high nitrites, or dissolved organic carbon (DOC) which enter the system via TDS. In essence, ORP is a compound indicator for multiple other factors in the system.

Chapter 3

Relevant Deep Learning Concepts

This chapter is divided into two parts. The first part presents some essential background on deep learning concepts which relate to the work done in this research and the components used to construct the model, which is presented in the second part of the chapter.

Section 3.1 presents background on deep anomaly detection, including the different neural network components that are used in this context and why they are applied. Of these, the autoencoder architecture in Section 3.1.1, convolutional neural networks in Section 3.1.2, recurrent neural networks (LSTM in particular) in Section 3.1.3, and scaled dot-product attention mechanism in Section 3.1.4. Other concepts covered in this section also include activation functions (Sigmoid, ReLU, and SELU in particular).

The model presented in this work, aMSCRED, is elaborated upon and contrasted with the model of Zhang et al. [1] on which it is based. Section 3.2 first gives an overview of the model, followed by examining each component individually and formalizing them mathematically. These components presented are the signature matrix generator, the convolutional encoder, the recurrent component, and the convolutional decoder. This is followed by the process to compare the input to the model and measuring the loss, used for training the model.

3.1 Anomaly Detection Concepts

A subclass of deep anomaly detection techniques, called Spatio-Temporal Networks (or STN) utilize a variety of traditional neural network layers, combined to create hybrid network architecture which learns both spatial and temporal dependencies present between pairs of timeseries in a given dataset. [4]

The aMSCRED model presented in Section 3.2 is no exception. It is built using the autoencoder architecture, utilizing a mixture of convolutional networks as the encoder-decoder pair with a recurrent neural network placed between them, acting on latent space of the convolution autoencoder.

Before delving into the details of the aMSCRED model, this section presents an overview of the fundamental components used. In the following subsections, a summary of terminology and basic operation of Autoencoder

networks, Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), and Scaled Dot-Product Attention mechanism are presented and contextualize why they are used, without getting into any formal mathematical rigor.

3.1.1 Autoencoder

The Autoencoder operates on the principle of learning to correctly recreate input data, given only a compressed representation of it in latent space, as depicted in Figure 3.1. They approximate the identity function

$$\hat{x} = f_{W,b}(x) \approx x \tag{3.1}$$

where x is the input, f is the model and W and b are the weights and biases of the network's layers. In principle, by compressing input data to a reduced dimensionality latent space, the network learns the underlying structure of the data, while disregarding superfluous information. In other words, information that has greater influence on correctly reconstructing inputs from the obtained latent space is kept, while the rest is discarded.[4]

Autoencoder loss, typically referred to as reconstruction loss, is given by,

$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\| \tag{3.2}$$

where x is the input and $\hat{x}$ is the reconstructed input.

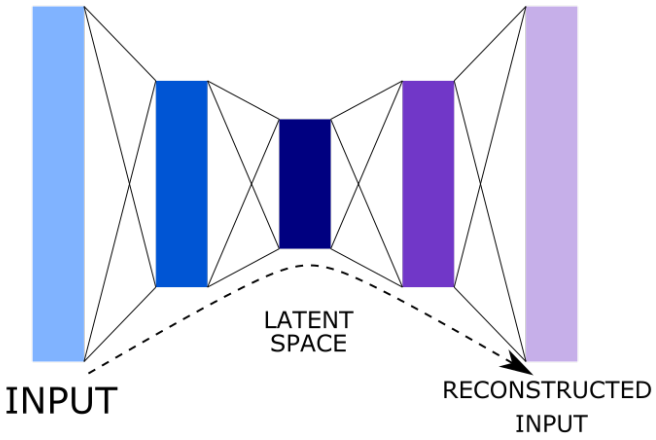

Figure 3.1: Simplified representation of the Autoencoder network architecture. Input x is first compressed into its Latent Space representation via the encoder; the decoder then learns to reconstruct the input ($\hat{x}$) as accurately as possible, given only the latent state.

3.1.2 CNN

The Convolutional Neural Network (CNN) is known for its pattern recognition capability while remaining resistant to small disturbances due to noise found in the input data.[9] Through repeated exposure to a pattern, the CNN gradually acquires knowledge of this pattern and learns to recognize it. This process happens in an unsupervised manner, meaning no prior knowledge of the existence of the pattern needs to be shared with the CNN.[9]

While a CNN is typically used with 2-dimensional image data (sometimes 3-dimensional if considering separate colour channels - RGB), it can be extended to higher dimensional image analogues, like a matrix or a tensor. (e.g.

the correlation matrix obtained from multivariate timeseries data).

When dealing with correlation matrices of timeseries data instead of image data, an additional property is gained, called shift-invariance. Shuffling the order of rows and columns (i.e. the ordering of timeseries in the correlation matrix) does not affect the outcome of the process.[1]

The mechanism through which a CNN functions is by sampling the input data m times with m kernels to obtain a stack of m filtered images. The dimensionality of the filtered space is reduced, via either max pooling or using strided samples (i.e. only sample every second position of the input (strides of 2×2), reducing filter image dimensionality by half). Multiple convolutional layers can be chained together, each layer's output feeding into the next layer's input. Normalization is typically achieved using ReLU activation function, due to its higher computational efficiency at the cost of a small increase of loss compared to tanh or sigmoid. The learnable parameters are the weights and biases of the kernels; these weights and biases are learned via the process of backpropagation.

To illustrate with a concrete example, consider a 32×32 pixel input image. The input data is thus a 32×32 matrix. Using 3×3 kernels and 2×2 strides, the kernels will output 16×16 filtered images (since the stride is 2 both in the x and y directions, leading to half the amount of pixels in both the x and y direction). This process is repeated for m kernels, each with its own weights and biases to be learned. The input data can thus be compressed an arbitrary amount in this fashion and the CNN learns patterns

in these lower dimensional spaces. This structured is illustrated in Figure 3.2.

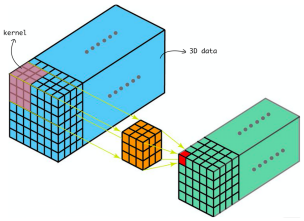

Figure 3.2: Example of a CNN on 3D dataset. The concept holds for any dimensional 32×32 input image is reduced to m 16×16 filtered images by m 3×3 kernels, using a stride size of 2. Figure from Shiva Verma's blog post, "Understanding 1D and 3D Convolution Neural Network", available at https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610 .

A key concept to remember when using CNN as the encoder and decoder of an autoencoder, is that data compressed via convolution is decompressed via the inverse operation, deconvolution. Deconvolution is the transpose of convolution. Higher dimensional images are created from smaller dimensional filtered images. Later in this work, the component of aMSCRED using this process is referred to as deconvolution or equivalently, as a convolutional decoder.

3.1.3 RNN

LSTM (Long Short-Term Memory) is a type of RNN (Recurrent Neural Network) capable of capturing long-term temporal dependencies in sequential data. Conceptually, it is composed of interconnected memory cells capable

of maintaining system state over time. This is accomplished using non-linear gates to control flow of information between interconnected cells (most commonly consisting of the input gate, output gate, forget gate, but other compositional variations exist).[10] Each cell receives as input, the output, and the hidden state of the previous cell. Internal computation through the gates of the cell updates the hidden state. The hidden state and output gate result are passed to the next cell in the chain, as shown in Figure 3.3.

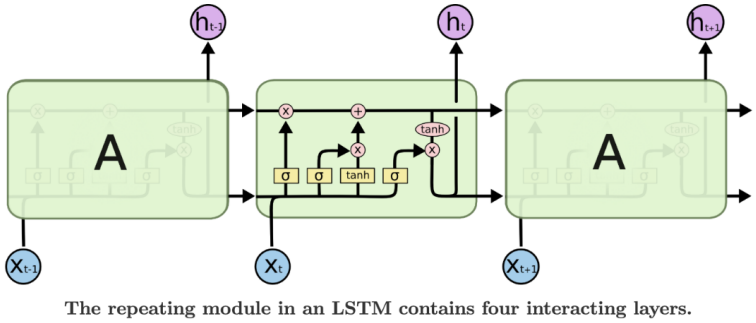

Figure 3.3: Visual representation of a LSTM network. Multiple interconnected cells learn temporal characteristics of the data. X_t is input sequence value at time t, h_t is the hidden state of cell t. Image from Christopher Olah's blog post *Understanding LSTM Networks*, (https://colah.github.io/posts/2015-08-Understanding-LSTMs/).

ConvLSTM is a variant of the LSTM such that the inputs, outputs, and hidden states of the recurrent cells are convolutional structures (i.e. the output of a CNN). In other words, it operates on higher dimensional tensors rather than one dimensional vectors, but otherwise the internal processes and gates behave identically. This variant of LSTM has shown, in previous work, the ability to learn spatiotemporal dependencies in multivariate timeseries

data.[11]

3.1.4 Scaled Dot-Product Attention

While LSTM models are able to capture patterns over a length of time dependent data, it does become increasingly computationally difficult and less accurate as the sequence length increases.

As such, the attention mechanism can be utilized with recurrent networks to model dependencies across time dependent data, regardless of the distance in time between data points. This effectively circumvents the limitation of LSTM by adaptively selecting timesteps which are most relevant to the current timestep.[1, 12]

Scaled Dot-Product Attention, the attention mechanism of interest in this work, is composed of the following matrix operations, as depicted in Figure 3.4.

The hidden state from each timestep in the ConvLSTM is element-wise multiplied with the output from the ConvLSTM (i.e. the state from the last timestep). These are scaled and summed and then normalized via softmax normalization, to yield a refined state of the ConvLSTM output.[12]

3.1.5 Activation Functions

There exist multiple activation functions that are utilized by different neural network types during model training. They are selected by judg-

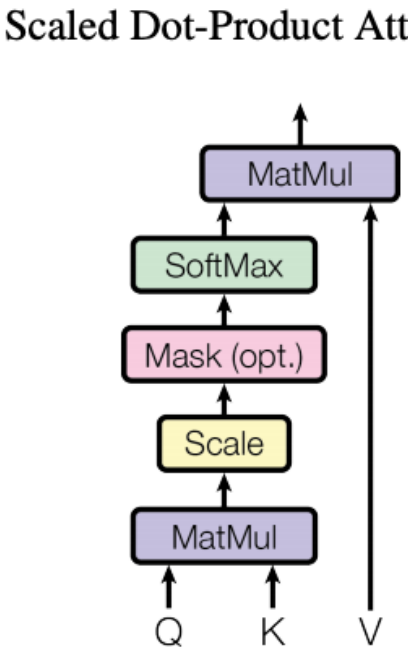

Figure 3.4: Scaled dot-product attention mechanism consists of element-wise matrix multiplication of Q and K, the states of interest. This is followed by a scaling step, and then normalization via SoftMax. The result is then matrix multiplied with the output state V, in order to weigh the elements that are most influential more favorably. Image from [12].

ing the tradeoff between accuracy of the trained model and computational complexity associated with the use of this activation function during back-propogation. The activation functions used by different layers of the model in this work are detailed briefly in this section, reasoning their choice by suitability for each application.

Sigmoid

The sigmoid function range is bound between 0 and 1, with limits approaching these values in the domain approach $-\infty$ and ∞ respectively, as shown in Figure 3.5, is given by :

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

Thus tends to be used when layer outputs predict probabilities, since it is bound between 0 and 1. A common place to find sigmoid activation functions is on the last layer of a network before the output, when dealing with probabilistic classification problems. It yields high accuracy on trained models but is computationally complex compared to pseudo-linear activation functions, such as ReLU. Sigmoid is used on the recurrent layers of the model presented in this work.

ReLU

ReLU, or rectified linear units, is a linear function for positive domain values, and 0 for all negative domain values, given by

$$f(x) = \max(0, x). \tag{3.4}$$

Its range is $[0, \infty)$. A plot of the ReLU function is shown in Figure 3.6. It is widely used today in CNNs due to its being less computationally expensive

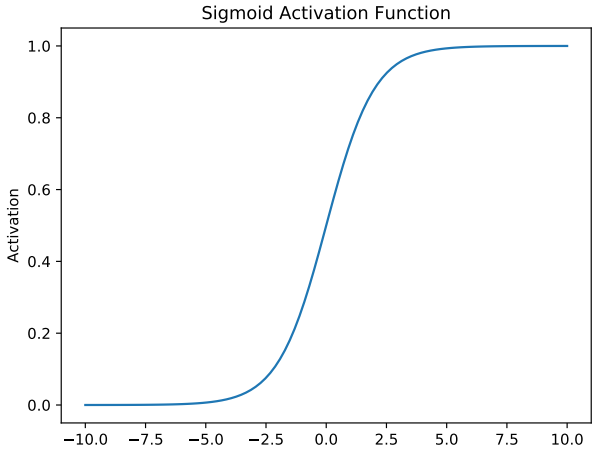

Figure 3.5: The sigmoid activation function is bound by 0 and 1 for all values of $x \in \mathbb{R}$.

as compared to sigmoid or tanh, yet does not yield a substantial decrease in model accuracy during training when dealing with convolutional structures. A noticeable drawback is that negative inputs are filtered out to zero, leading to loss of information when negative values are input into layers utilizing this activation function.

To remediate the drawback of ReLU associated with negative values being rectified, SELU is used, which has a non-zero assignment for negative values.

SELU

Scaled Exponential Linear Units is a newer variant of the ReLU activation function, where the negative elements in the domain are scaled exponentially

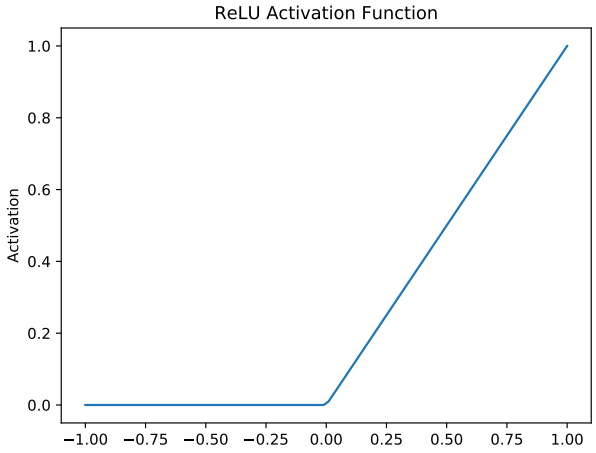

Figure 3.6: The Rectified Linear Unit (ReLU) activation function.

by some factor, α. It is given by

$$SELU(x) = \lambda \begin{cases} x, \text{if} x > 0 \\ \alpha e^x - \alpha, \text{otherwise} \end{cases} \tag{3.5}$$

which is plotted in Figure 3.7.

SELU is self-normalizing and thus the network using it will tend to converge faster than other activation functions. This property also makes the vanishing and exploding gradient problem disappear.[13] For this reason it is increasingly being adopted in network activation functions. SELU is the activation function used for training CNN layers in this work.

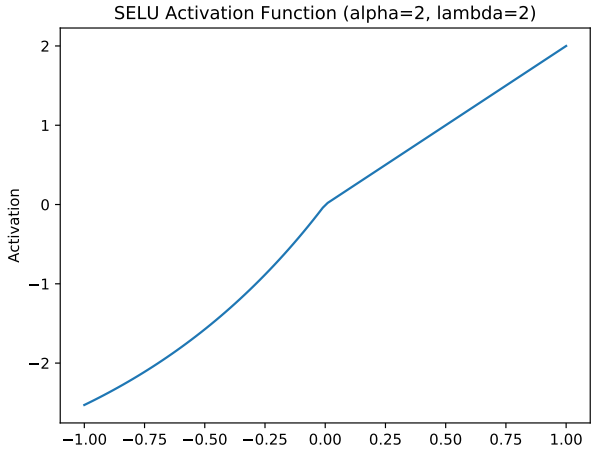

Figure 3.7: The Scaled Exponential Linear Unit (SELU) activation function. In this example, for $\alpha = 2$ and $\lambda = 2$.

3.2 aMSCRED Model

The adaptive-Multi-Scale Convolutional Recurrent Encoder-Decoder (aMSCRED) model is presented in this section as a candidate technique to perform anomaly detection and root cause identification of multivariate time-series data obtained from a recirculating aquaculture system (RAS). It builds upon the MSCRED model, which has shown in previous work to have superior anomaly detection performance to traditional, non-deep learning, approaches. [1]

The "adaptive" variant presented in this work modifies certain components to be flexible in their choice of strategies to employ, which can be adapated by empirically derived model performance in an automated fashion, via hyperparameter tuning that can be done on a per-dataset basis. The

most notable of these adaptive components have the ability to dynamically shape the CNN structure, dynamically shape the RNN structure, selectively enable the attention mechanism in the recurrent component of the model, and select the appropriate correlation function to use while generating signature matrices from timeseries data.

3.2.1 Component Overview

The aMSCRED is categorized as a Spatial Temporal Network (STN), which is a class of techniques that learns both spatial and temporal features by employing a hybrid mixture of neural networks, arranged in an autoencoder architecture; in particular, spatial features are learnt via CNN layers and temporal features via ConvLSTM layers, optionally augmented with scaled dot-product attention. [1, 4]

The multiple components that characterize aMSCRED (and MSCRED) and their purpose are summarized as follows.

First, signature matrices are generated from the multivariate timeseries data. Signature matrices are the data on which the model learns. This process is a one-way operation. The timeseries data cannot be reconstructed from the signature matrices. The signature matrices represent the pairwise correlation between each pair of timeseries during a window of time. The signature matrix at timestep t, written as M^t, is a snapshot of the pairwise correlation of each timeseries at time t for a duration of s windows, with sizes w_1, w_2, w_3. Using many different window sizes yields a multi-scale signature

matrix.

Next, sequential signature matrices are assembled into groups of size h. This is denoted as the set of multi-scale signature matrices, $M^{t-h}, M^{t-h+1}, \cdots, M^{t}$. One of these sets is a single input for the model. This set is encoded with a Convolutional Encoder, composed of l stacked convolutional layers, which compresses the set of h signature matrices into a large collection of smaller dimensional samples.

The h outputs from each convolutional are fed into their own ConvLSTM layer, optionally augmented with a scaled dot-product attention mechanism, which learns temporal patterns present in the compressed spatial data.

The last (i.e. most recent timestep) of the h outputs from each recurrent layer are then concatenated with the output from the next layer's Convolutional Decoder and fed into its own Convolutional Decoder (except for the $l-$th layer, in which only the RNN output itself is considered). The objective of the convolutional decoder is to reconstruct the $h-$th input signature matrix as accurately as possibly, called the reconstructed matrix, $\hat{M}^t$.

The residual matrix, R^t, at t, is obtained by subtracting M^t and $\hat{M}^t$. This matrix represents the element-wise reconstruction loss. The objective of the model is to minimize the reconstruction loss during training, i.e. from M^t, the model should output $\hat{M}^t$ such that each entry in R^t is as close to 0 as possible. This completes the autoencoder structure.

This structure is illustrated in Figure 3.8. For brevity, the model depicted uses $l = 4$ layers only, however this structure can accomodate an arbitrarily

large number of layers. In the orange box (1), a set of h sequential signature matrices is grouped. This set is input into $l = 4$ layer convolutional encoder, in the red box (2), denoted as Conv1, Conv2, Conv3, Conv4, respectively.

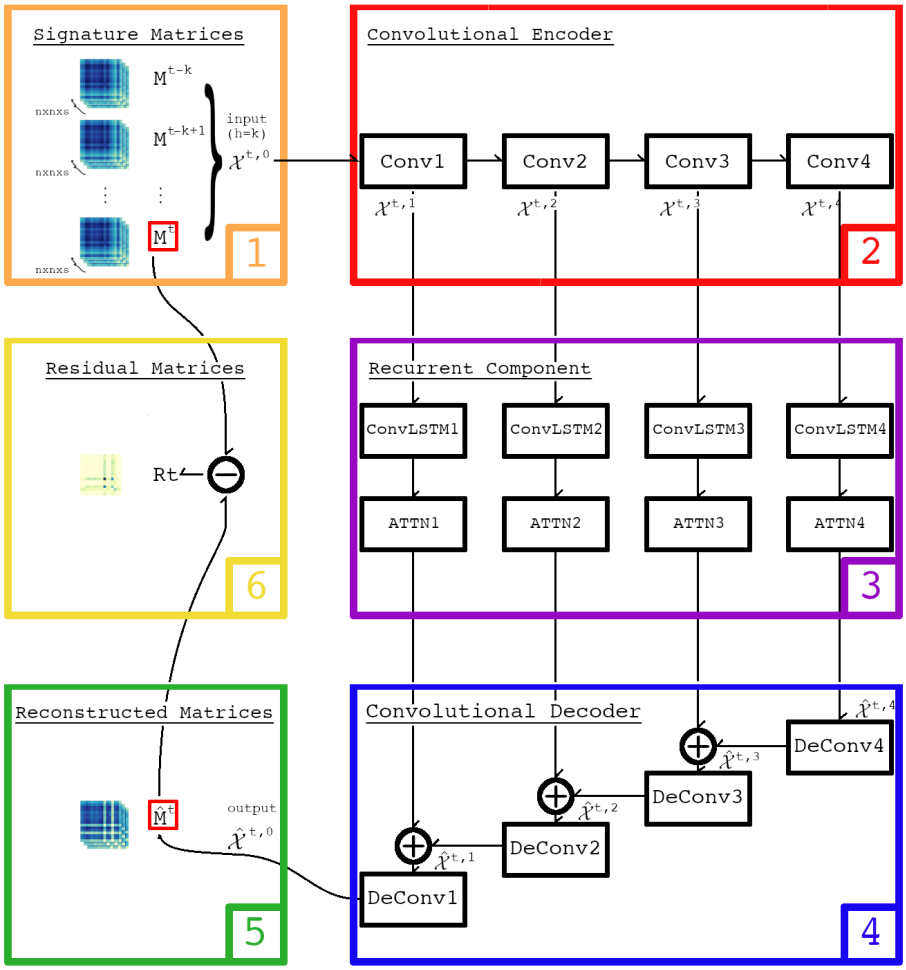

Figure 3.8: Overview of aMSCRED architecture. Composed of a recurrent network (purple) acting on the latent space of a convolutional encoder-decoder pair (red and blue, respectively).

The convolutional layers are chained, i.e. output from layer k is used

as the input of layer $k+1$ and so on. In the purple box (3), the outputs from each convolutional layer is input into its respective ConvLSTM layer, labelled ConvLSTM1, ConvLSTM2, ConvLSTM3, and ConvLSTM4, respectively. Each ConvLSTM layer returns a sequence of h sequential outputs, which are fed through their own scaled dot-product attention mechanism, labelled Attn1, Attn2, Attn3, and Attn4, respectively. The attention layer consumes all but the last of the h timesteps, which becomes the output. In the blue box (4), the deconvolutional layers, labelled DeConv1, DeConv2, DeConv3, and DeConv4, reconstruct the signature matrix, from the outputs of the recurrent layers. Starting from $k = l$ and working backwards, the output of the $k-$th attention layer is concatenated with the output of the $(k-1)-$th DeConv layer. The result is input into the $k-$th DeConv layer. The only exception to this is for the DeConv4 layer, for which only the output of Attn4 is used since there is no DeConv layer output beneath it. This process is repeated until the final ($l = 1$) layer is attained, which outputs the reconstructed signature matrix, shown in the green box (5). In the yellow box (6), the residual matrix R^t is computed by taking the difference between the last input signature matrix (M^t) and the reconstructed matrix $\hat{M}^t$. The loss is measured by computing the norm of the residual matrix.

These components are explained in further detail, with mathematical rigor, in the following subsections.

3.2.2 Signature Matrices

Generating signature matrices is the first step in the aMSCRED architecture, represented by the orange box of Figure 3.9.

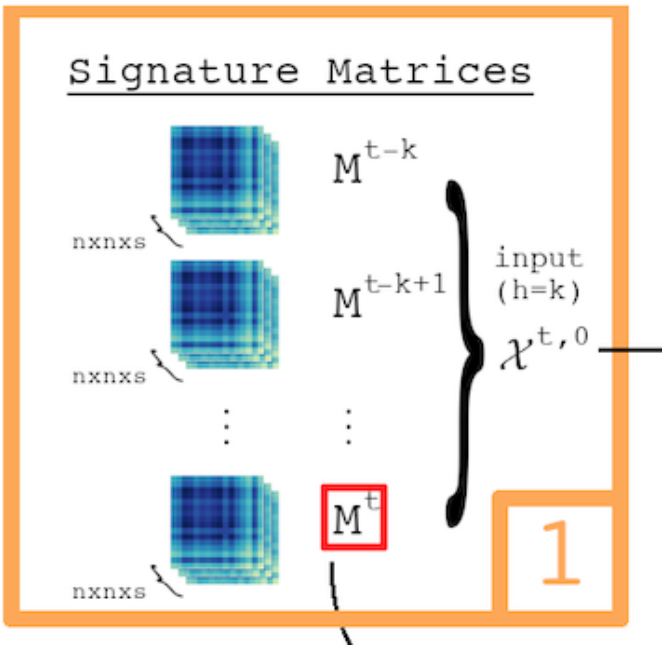

Figure 3.9: Signature matrices are generated in groups of k sequential matrices, yielding a single input for model, the tensor $\mathcal{X}^{t,0}$.

Signature matrices are computed from multivariate timeseries, $\boldsymbol{X}$, data via the following process. This process is uni-directional, meaning the timestep of the original dataset cannot be reconstructed from the signature matrix representation.

Let $\boldsymbol{X}$, given by

$$\boldsymbol{X} = (\boldsymbol{x_1}, ..., \boldsymbol{x_n})^T \in \mathbb{R}^{n \times T} \tag{3.6}$$

denote historical data of n time series of length T.

Signature matrices, M^t, are $n \times n$ matrices that characterize the state of the multivariate timeseries at time step t. They are computed by taking the pairwise inner-product of n^2 pairs obtained from n timeseries that compose the system, over a window of length w, i.e. from timestep $t - w$ to t.

Constructing the signature matrix, M^t, in this fashion captures only correlations between values of the timeseries, while ignoring the values individually. In essence these matrices represent how values in all timeseries tend to change together. This gives the signature matrices the property of robustness to noise in the input signal.[1]

Given any two series, $\boldsymbol{x_i^w} \in \boldsymbol{X^w}$ and $\boldsymbol{x_j^w} \in \boldsymbol{X^w}$, in the timeseries segment $\boldsymbol{X^w}$, where each segment is given by

$$\begin{aligned} \boldsymbol{x_i^w} &= (x_i^{t-w}, x_i^{t-w-1}, \cdots, x_i^t) \\ \boldsymbol{x_j^w} &= (x_j^{t-w}, x_j^{t-w-1}, \cdots, x_j^t) \end{aligned} \tag{3.7}$$

their pairwise correlation for the segment, m_{ij}^t, is given by,

$$m_{ij}^t = f_c(x_i^t, x_j^t) \tag{3.8}$$

where $f_c(\bullet)$ is the correlation function and $m_{ij}^t \in M^t$ are the resulting $i - j -$ th element of the signature matrix M^t. Thus, the signature matrix is

represented as

$$M^t = \begin{bmatrix} m_{11}^t & \cdots & m_{1n}^t \\ \vdots & \ddots & \vdots \\ m_{n1}^t & \cdots & m_{nn}^t \end{bmatrix} \tag{3.9}$$

The correlation function, f_c, used in Zhang et al. is given by

$$m_{ij}^t = f_c(x_i^t, x_j^t) = \frac{\sum_{\delta=0}^{w} x_i^{t-\delta} x_j^{t-\delta}}{\kappa} \tag{3.10}$$

where κ is a rescale factor (in this case, $\kappa = w$).[1]

This correlation function, referred to herein as constant time, scales correlation coefficients by the window size, but does not weigh based on temporal distance. Values further away in time impact the computed correlation as much as most recent values. Furthermore, this correlation function is not invariant under linear transformations on individual timeseries, which could negatively impact the learning process by confusion of important features.

A novel modification introduced in aMSCRED is the ability to adaptively select from a set of predefined correlation functions, defined below. The purpose of this functionality is to empirically determine which correlation function leads to the best anomaly detection performance on a per-dataset basis during hyperparameter tuning.

The first proposed correlation function is the Pearson correlation coeffi-

cient, given by

$$m_{ij}^t = \frac{\sum_{k=0}^{w}(x_i^{t-k} - \bar{x}_i^t)(x_j^{t-k} - \bar{x}_j^t)}{\sqrt{\sum_{k=1}^{T}(x_i^k - \bar{x}_i)^2 \sum_{k=1}^{T}(x_j^k - \bar{x}_j)^2}} \tag{3.11}$$

where $\bar{x}_i^t$ denotes the mean value of timeseries segment x_i over period $t-k$ to t, and T is the set of all timesteps in the timeseries.

These coefficients, while similar to the constant time coefficients, are additionally invariant under linear transformations, which may improve learning conditions by reducing feature confusion. [3]

The two other proposed correlation function are called fractional time and exponential time correlation function, respectively. These coefficients are weighed inversely to window distance, that is,

$$m_{ij}^t = \frac{\sum_{\delta=0}^{w} W_\delta^t x_i^{t-\delta} x_j^{t-\delta}}{\kappa} \tag{3.12}$$

where W_δ^t are the weights associated to distance δ from timestep t.

Two types of time weighting are implemented in aMSCRED : fractional and exponential. Fractional assigns static weights $\frac{1}{k+1}$ for $k \in \{1, 2, ..., w\}$, Exponential assigns static weights $\frac{1}{2^k}$ for $k \in \{1, 2, ..., w\}$. The MSCRED scaled constant time and Pearson function above can be thought of as having static weights of 1 for all k. Thinking of it in terms of this analogue, the weight in each previous timesteps with respect to distance to the current timestep can be plotted, as shown in Figure 3.10. Here it is clear that the

weight of previous timesteps decay much quicker for the exponential weighted correlation function than the fractional, while the other two have constant relative weight of 1.

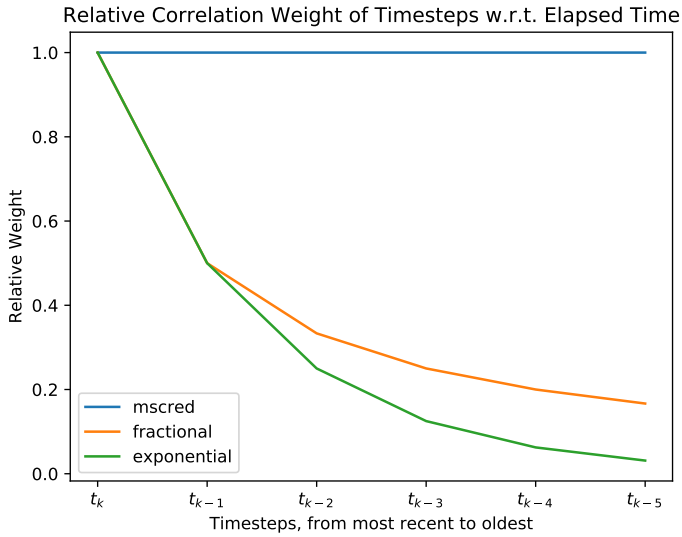

Figure 3.10: Relative correlation weights for timesteps with respect to distance in time. Older timesteps carry less weight on correlation coefficient at current timestep. The exception are constant time (MSCRED) and Pearson, which use a constant weight of 1 (i.e. independent of distance in time).

The purpose of adding weights to the correlation coefficients is to create another degree of freedom in which the model can learn importance of previous measurements with respect to current measurements.

While the correlation functions tested in this work are all assigned static weights, future work could implement dynamic weights which are learned via backpropagation while training the model. More will be said about this in Section 5.6.

In order to create the multi-scale signature matrices required by model, multiple signature matrices are computed for different windows sizes, w, in order to gather information on pairwise correlation at different scales. The number of scales, or windows, is denoted by s. This yields a $n \times n \times s$ signature matrix, as shown in Figure 3.11.

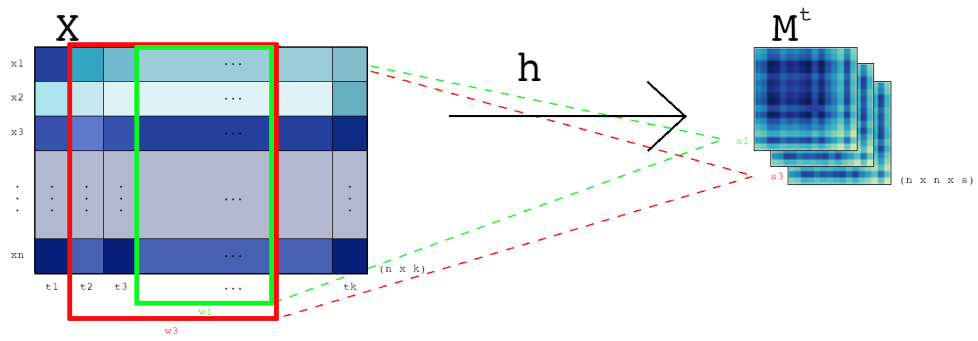

Figure 3.11: M^t is the $n \times n \times s$ signature matrix generated from multivariate timeseries, $\boldsymbol{X}$. Each of the s channels in the signature matrix is generated by applying the correlation function, h, over different window sizes, w.

3.2.3 Convolutional Encoder

The convolutional encoder utilizes a chain of convolutional layers, each encoding wide-and-short spatial patterns found in the signature matrices, to thin-and-long reduced dimensionality latent space. Here, wide and thin refers to the dimensionality of the square matrices ($n \times n$ being the widest, and it gets sampled down to thinner dimensions $\frac{n}{2} \times \frac{n}{2}$, $\frac{n}{4} \times \frac{n}{4}$, and so on) and short and long refers to the number of channels, i.e. $s = 3$ for the input

signature matrices channels is short, whereas the number of filtered images (number of kernels) is longer after convolution (e.g. 256 kernels yield 256 filtered images / channels).

The input to the convolutional encoder is a tensor composed of a stack of h sequential signature matrices, $\mathcal{X}^{t,0} \in \mathbb{R}^{h \times n \times n \times s}$, where t is the timestep at which the s signature matrices were sampled, and $l = 0$ references the layer number (0 since it is the input). The input is fed sequentially into a chain of four convolutional layers, labelled Conv1 through Conv4, as shown in Figure 3.12. At each step, the output of the current convolutional layer is both stored for the next step and fed into the input of the next convolutional layer.

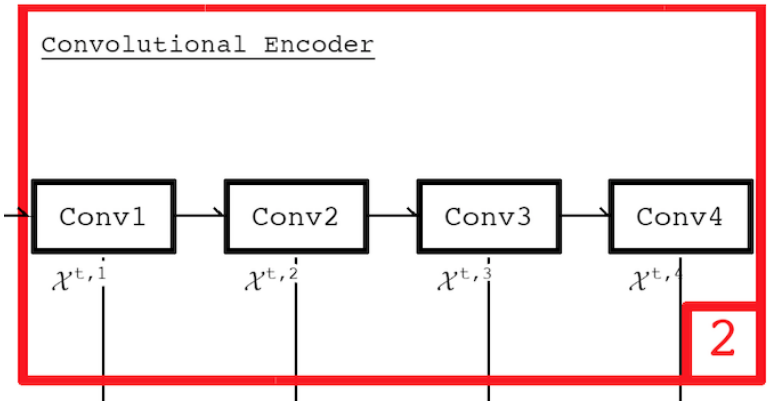

Figure 3.12: Convolutional encoder are chained l levels deep, with the output from each layer being fed into the input of the next layer. The outputs from each layer are then gathered and fed into the inputs of their respective layers in the next component of the aMSCRED model.

Given that $\mathcal{X}^{t,l-1} \in \mathbb{R}^{n_{l-1} \times n_{l-1} \times d_{l-1}}$ denotes the features maps from the

previous, $(l-1)-$th, layer, the output of the $l-$th layer is given by

$$\mathcal{X}^{t,l} = f(W^l * \mathcal{X}^{t,l-1} + b^l) \tag{3.13}$$

where $*$ is the convolution operator, $f(\bullet)$ is the activation function, $W^l \in \mathbb{R}^{k_l \times k_l \times d_{l-1} \times d_l}$ are the d_l convolutional kernels of size $k_l \times k_l \times d_{l-1}$, $b^l \in \mathbb{R}^{d_l}$ is a bias term, and $\mathcal{X}^{t,l} \in \mathbb{R}^{n_l \times n_l \times d_l}$ is the output feature map at $l-th$ layer. SELU is chosen as the activation function for all convolutional layers.

In the implementation by Zhang et al., the authors found the structure presented in Table 3.1 worked best for their use-case.[1] In this work, the structure of convolutional layers in aMSCRED are implemented as tunable hyperparameters which are tuned on a per-dataset basis. These hyperparameters are presented later, in Section 4.4.

Table 3.1: Convolutional encoder network structure from the implementation of Zhang et al. [1]

Layer	**Kernels**	**Kernel Size**	**Stride**
Conv1	32	$3 \times 3 \times 3$	1×1
Conv2	64	$3 \times 3 \times 32$	2×2
Conv3	128	$2 \times 2 \times 64$	2×2
Conv4	256	$2 \times 2 \times 128$	2×2

3.2.4 Attention-Based ConvLSTM

As explained in the previous subsection, the convolutional encoder generates spatial feature maps which are dependent on previous time steps, due

to the nature of timeseries.

A ConvLSTM network acts on the outputs of the convolutional encoder in order to learn temporal dependencies of the spatially encoded timeseries data. However, the performance of this type of network deteriorates as distances across the time axis is increased (distance being the number of time steps forward or backwards in time). As the length of the sequences or temporal pattern increases, performance deteriorates.[1, 11]

In the MSCRED model, a temporal scaled dot-product attention mechanism is implemented to act on the outputs of the ConvLSTM layer, which acts as a means to adaptively select previous timesteps that are most relevant to the current timestep. Thus, the Attention-Based ConvLSTM models a joint representation of spatial patterns and temporal information found in each group of signature matrices.[1]

Given feature map, $\mathcal{X}^{t,l}$, of the l-th convolutional layer and previous hidden state $\mathcal{H}^{t-1,l} \in \mathbb{R}^{n_l,n_l,d_l}$, the current hidden state, $\mathcal{H}^{t,l} \in \mathbb{R}^{n_l,n_l,d_l}$, is updated via the ConvLSTM($\bullet$)

$$\mathcal{H}^{t,l} = \text{ConvLSTM}(\mathcal{X}^{t,l}, \mathcal{H}^{t-1,l}). \tag{3.14}$$

The ConvLSTM(•) update function is defined as follows :

$$\boldsymbol{z}^{t,l} = \sigma(\tilde{W}^{l}_{\mathcal{XZ}} * \mathcal{X}^{t,l} + \tilde{W}^{l}_{\mathcal{HZ}} * \mathcal{H}^{t-1,l} + \tilde{W}^{k}_{\mathcal{CZ}} \circ \mathcal{C}^{t-1,l} + \tilde{b}^{l}_{\mathcal{Z}}) \tag{3.15}$$

$$\boldsymbol{r}^{t,l} = \sigma(\tilde{W}^{l}_{\mathcal{XR}} * \mathcal{X}^{t,l} + \tilde{W}^{l}_{\mathcal{HR}} * \mathcal{H}^{t-1,l} + \tilde{W}^{l}_{\mathcal{CR}} \circ \mathcal{C}^{t-1,l} + \tilde{b}^{l}_{\mathcal{R}}) \tag{3.16}$$

$$\mathcal{C}^{t,l} = \boldsymbol{z}^{t,l} \circ \tanh(\tilde{W}^{l}_{\mathcal{XC}} * \mathcal{X}^{t,l} + \tilde{W}^{l}_{\mathcal{HC}} * \mathcal{H}^{t-1,l} + \tilde{b}^{l}_{\mathcal{C}}) + \boldsymbol{r}^{t,l} \circ \mathcal{C}^{t-1,l} \tag{3.17}$$

$$\boldsymbol{o}^{t,l} = \sigma(\tilde{W}^{l}_{\mathcal{XO}} * \mathcal{X}^{t,l} + \tilde{W}_{\mathcal{HO}} * \mathcal{H}^{t-1,l} + \tilde{W}_{\mathcal{CO}} \circ \mathcal{C}^{t,l} + \tilde{b}^{l}_{\mathcal{O}}) \tag{3.18}$$

$$\mathcal{H}^{t,l} = \boldsymbol{o}^{t,l} \circ \tanh(\mathcal{C}^{t,l}) \tag{3.19}$$

where $*$ is the convolutional operator, $\circ$ is the Hadamard product, σ is the sigmoid function, $\tilde{W}^{l}_{\mathcal{XZ}}$, $\tilde{W}^{l}_{\mathcal{HZ}}$, $\tilde{W}^{l}_{\mathcal{CZ}}$, $\tilde{W}^{l}_{\mathcal{XR}}$, $\tilde{W}^{l}_{\mathcal{HR}}$, $\tilde{W}^{l}_{\mathcal{CR}}$, $\tilde{W}^{l}_{\mathcal{XC}}$, $\tilde{W}^{l}_{\mathcal{HC}}$, $\tilde{W}^{l}_{\mathcal{XO}}$, $\tilde{W}^{l}_{\mathcal{HO}}$, $\tilde{W}^{l}_{\mathcal{CO}} \in \mathbb{R}^{\tilde{k}_l \times \tilde{k}_l \times \tilde{d}_l \times \tilde{d}_l}$ are $\tilde{d}^{l}$ convolutional kernels of size $\tilde{k}_l \times \tilde{k}_l \times \tilde{d}_l$, and $b^{\tilde{l}}_{\mathcal{Z}}$, $b^{\tilde{l}}_{\mathcal{R}}$, $b^{\tilde{l}}_{\mathcal{C}}$, $b^{\tilde{l}}_{\mathcal{O}} \in \mathbb{R}^{\tilde{d}_l}$ are bias terms for the $l - th$ layer in the ConvLSTM.[1, 11] $\mathcal{X}^{t,l}$ denotes cell inputs, $\mathcal{C}^{t,l}$ denotes cell outputs. $\boldsymbol{z}^{t,l}$, $\boldsymbol{r}^{t,l}$, and $\boldsymbol{o}^{t,l}$ denote the input, forget, and output gates, respectively.

The temporal attention mechanism is subsequently applied to the ConvLSTM outputs. The attention mechanism presented in MSCRED is a simplified version of the model proposed by the original paper from [1, 14]. It learns to select which time steps are most relevant to the current time step, in order to form refined output feature maps, $\hat{\mathcal{H}}^{t,l}$, given by

$$\hat{\mathcal{H}}^{t,l} = \sum_{i \in (t-h,t)} \alpha^i \mathcal{H}^{i,l} \tag{3.20}$$

$$\alpha^i = \frac{\beta^i}{\sum_{i \in (t-h,t)} \beta^i} \tag{3.21}$$

$$\beta^i = \exp \frac{Vec(\mathcal{H}^{t,l})^T \} Vec(\mathcal{H}^{i,l})}{\chi} \tag{3.22}$$

where $Vec(\bullet)$ is the vectorized representation of the corresponding tensor-slice, χ is a rescale factor.

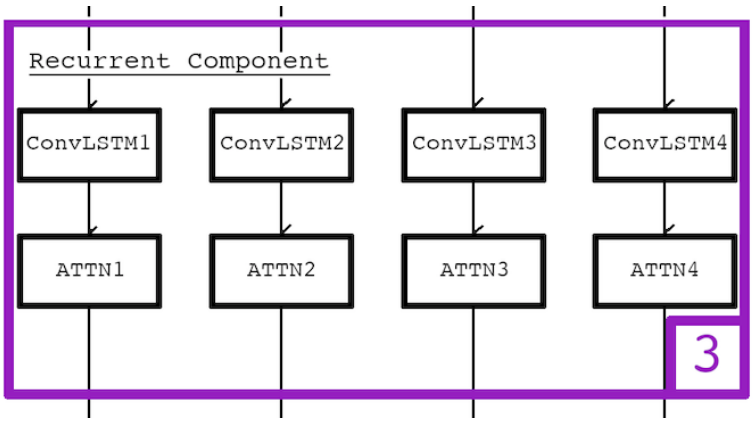

Figure 3.13: Outputs from each layer l of the convolutional encoder are fed into the respective ConvLSTM layer. The output sequences from each ConvLSTM is passed to its respective attention layer. These outputs are pushed forward to the convolutional decoder.

In this work, aMSCRED includes a novel feature to selectively use attention or to bypass it altogether on a per-dataset basis. This is determined during hyperparameter tuning, based on which choice leads to best anomaly detection performance results. The attention bypass layer simply feeds for-

ward the last output of the ConvLSTM layer's h outputs to the next layer. Figure 3.14 depicts the differences between the attention layer and the attention bypass layer.

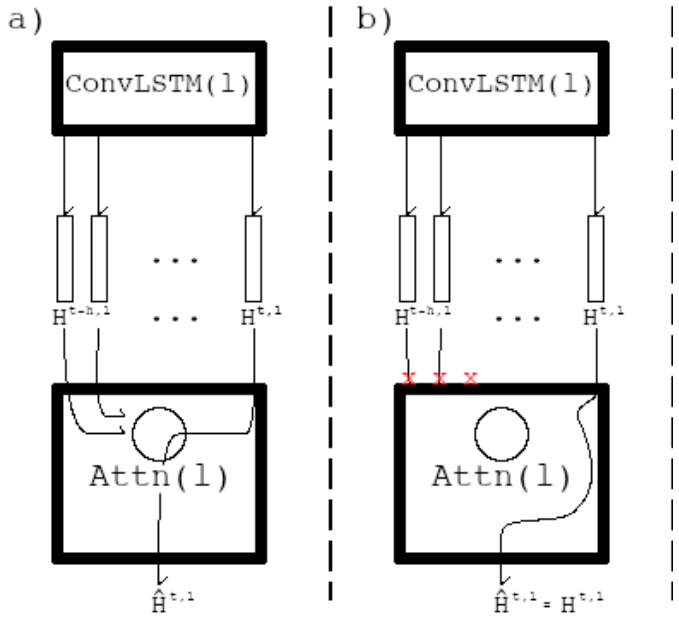

Figure 3.14: (a) Each hidden state, $\mathcal{H}^{t,l}$ from the ConvLSTM of layer l is passed into the attention layer yielding refined feature map $\hat{\mathcal{H}}^{t,l}$. (b) the attention layer is bypassed; only the last hidden state, $\mathcal{H}^{t,l}$, of ConvLSTM at layer l is fed forward unmodified. The remaining hidden states are discarded.

Other tunable hyperparameters for the ConvLSTM component of the model are the lookback distance, h, timesteps for the ConvLSTM to consider while learning. The rescale factor, χ is set h, as in the MSCRED implementation, but this could be modified to a variable weight parameter in future work.

3.2.5 Convolutional Decoder

The convolutional decoder attempts to reconstruct original signature matrices as faithfully as possible using only outputs of the previous recurrent component, i.e. the feature maps, $\hat{\mathcal{H}}^{t,k}$, obtained for each of the $k \in 1, 2, ..., l$ layers. The refined feature map of the $(k+1)$-th ConvLSTM layer $\hat{\mathcal{H}}^{t,k+1}$ is concatenated with the output of the $(k+1)$-th deconvolutional layer $\hat{\mathcal{X}}^{t,k+1}$. The concatenated result is input into the k-th deconvolution layer, to obtain the output of the k-th layer, $\hat{\mathcal{X}}^{t,k}$. This process is repeated similarly, for all k in $1, 2, ..., l$. The exception to this rule is at the l-th layer, only the feature map of the l-th ConvLSTM layer, $\hat{\mathcal{H}}^{t,l}$, is used as input for the deconvolution layer, since there is no output from a previous layer to concatenate it with. According to the results of previous work, concatenating ConvLSTM feature maps with the output of the previous DeConv layer in this manner was shown to improve anomaly detection performance.[1] This process, depicted in Figure 3.15, is formalized as

$$\hat{\mathcal{X}}^{t,l-1} = \begin{cases} f(\hat{W}^{t,l} \circledast \hat{\mathcal{H}}^{t,l} + \hat{b}^{t,l}), & \text{when l = 4} \\ f(\hat{W}^{t,l} \circledast [\hat{\mathcal{H}}^{t,l} \oplus \hat{\mathcal{X}}^{t,l}] + \hat{b}^{t,l}), & \text{when l = 3,2,1} \end{cases} \tag{3.23}$$

where $\circledast$ is the deconvolution operator, $\oplus$ is the concatenation operator, $f(\bullet)$ is the activation function, $\hat{W}^{l} \in \mathbb{R}^{\hat{k}_l \times \hat{k}_l \times \hat{d}_l \times \hat{d}_{l-1}}$ are the filter kernels of the l-th deconvolutional layer, and $\hat{b}^{l} \in \mathbb{R}^{\hat{d}_l}$ are bias parameters (of each of the l deconvoluation layers). $\hat{\mathcal{H}}^{t,l}$ is the output from the recurrent component

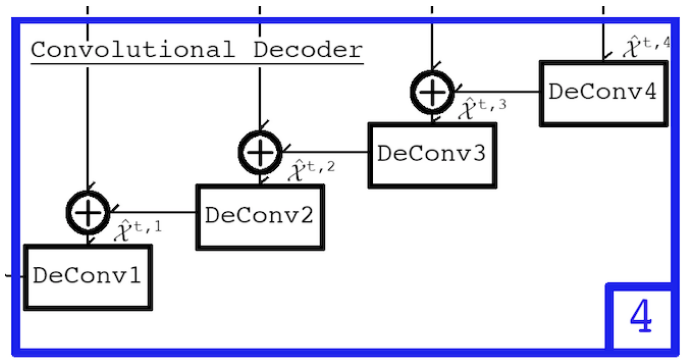

Figure 3.15: The $k-$th convolutional decoder layer, DeConv(k), receives as input the concatenation of the output of DeConv($k+1$) and output from ConvLSTM($k+1$). Its output is concatenated with the output from ConvLSTM(k) and are input into the next DeConv layer. This process repeats until the reconstructed signature matrix is obtained, $\hat{M}^t$, at the $0-$th layer.

at layer l, and $\hat{\mathcal{X}}^{t,l}$ is the output of the l-th deconvolutional layer.

The output at the final layer, $\hat{\mathcal{X}}^{t,0} \in \mathbb{R}^{n \times n \times s}$, is a $n \times n \times s$ matrix called the reconstructed matrix. Unlike the input tensor, $\mathcal{X}^{t,0} \in \mathbb{R}^{h \times n \times n \times s}$, which is composed of a stack of h signature matrices, this reconstructed matrix represents only the last of the h original signature matrices that composed the input tensor, shown in Figure 3.16.

Note, the activation function used during deconvolution matches the activation function used in the convolutional encoder (SELU).

The structure of the deconvolutional layers, labelled DeConv4 through DeConv1, following order of execution, is given in Table 3.2. The structure in this table relates to the implementation from Zhang et al. [1] and is just used here as a stand-in to concretely illustrate the model structure. In the aMSCRED model, this structure is left as a set of tunable hyperparameters.

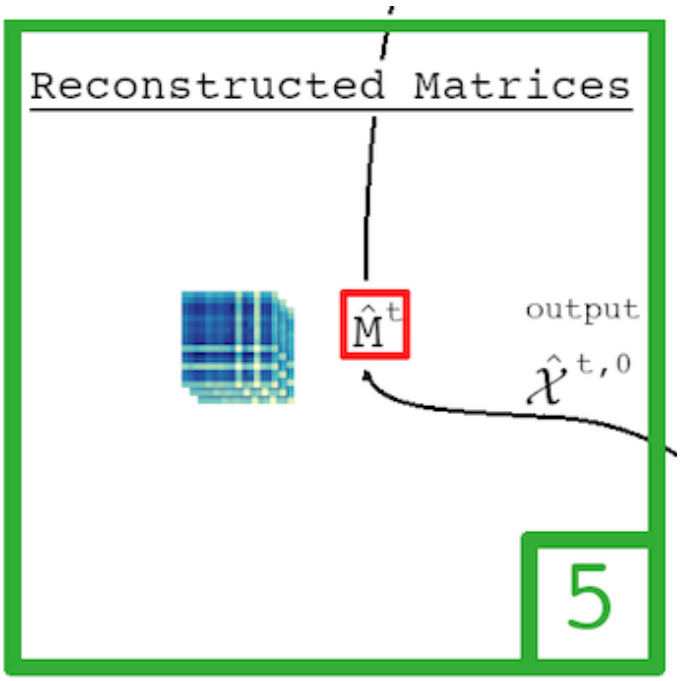

Figure 3.16: The reconstructed matrix, $\hat{M}^t$, is constructed by the trained model, attempting to as accurately as possible reproduce the $h-$th input signature matrix from step 1.

Specifically, the number of kernels at each layer, the kernel sizes at each layer, and the kernel strides at each layer are tunable. The full set of the hyperparameters tested in this work are presented in Section 4.4.

3.2.6 Loss Function

After obtaining the reconstructed signature matrices from the convolutional decoder, the difference with respect to the original signature matrix is computed, yielding the residual matrix. The residual matrix is just the difference between the last (most recent) input signature matrix and the

Table 3.2: Convolutional decoder component structure. These values refer to the ones found in the implementation of Zhang et al.[1] A full list of tested hyperparameter are presented in Section 4.4.

Layer	# Kernels	Kernel Size	Stride
DeConv4	128	$2 \times 2 \times 256$	2×2
DeConv3	64	$2 \times 2 \times 128$	2×2
DeConv2	32	$3 \times 3 \times 64$	2×2
DeConv1	3	$3 \times 3 \times 64$	1×1

reconstructed matrix, as seen in Figure 3.17.

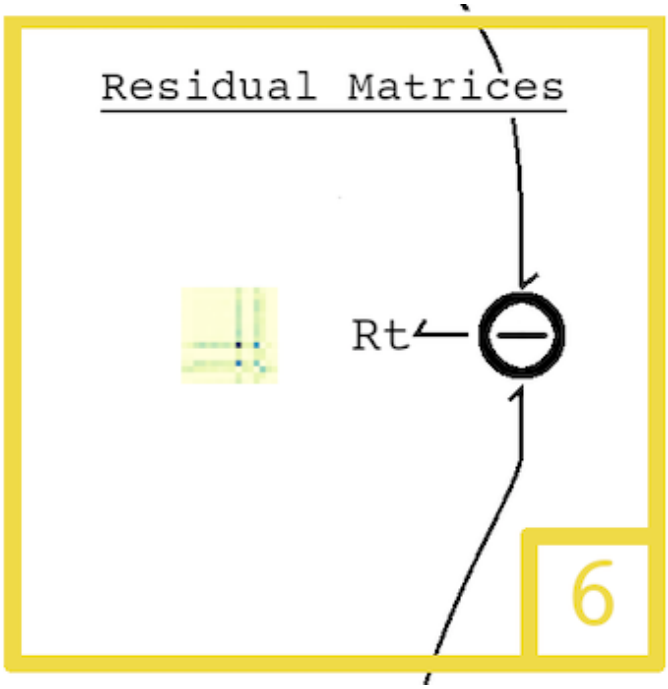

Figure 3.17: The residual matrix, R^t, is obtained by taking the difference between the reconstructed matrix $\hat{M}^t$ and the last (most recent) input signature matrix, M^t.

The residual matrix is then used to compute the loss, in this case also called the reconstruction error. The loss function of the residual matrix is

given by

$$\mathcal{L}_{\text{MSCRED}} = \sum_{t} \sum_{c=1}^{s} \left\| \mathcal{X}_{:,:,c}^{t,0} - \hat{\mathcal{X}}_{:,:,c}^{t,0} \right\|_F^2 \tag{3.24}$$

where $\mathcal{X}_{:,:,c}^{t,0} \in \mathbb{R}^{n \times n}$ is a single channel (scale) of the input system $n \times n$ signature matrix obtained in the first step and $\hat{\mathcal{X}}_{:,:,c}^{t,0} \in \mathbb{R}^{n \times n}$ is its respective reconstructed signature matrix obtained. The difference $\mathcal{X}_{:,:,c}^{t,0} - \hat{\mathcal{X}}_{:,:,c}^{t,0}$ yields the residual matrices. The reconstruction error, or loss, is the Frobenius norm of elements of the residual matrix.

This loss function is used during training to gauge how well the model is able to reconstruct signature matrices, given normal input signature matrices. The loss function is not considered as a metric to measure anomaly detection performance in this work, since its main purpose is to measure the model's ability to reconstruct normal data, given normal data. During training, only normal data is used to train the model, thus there is no anomalous data present in the training dataset. The marker of a well-trained model would need to poorly reconstruct signature matrices with anomalous data. However, models with lower reconstruction loss do tend to score better in theory; a smaller reconstruction loss on normal data creates a larger signal to noise ratio between anomalous data and normal data. This could lead to a more noticeable reconstruction error when anomalous data is fed into the model.

Chapter 4

Methods

This chapter's focus is to elaborate upon the methods used to train the model, as well as methods undertaken to measure the anomaly detection performance of the trained model in the context of the RAS dataset.

Starting in Section 4.1, the facility where the research is conducted is introduced, including a description of the RAS system design, the sensors it utilizes, and the monitoring system which was in place prior to this research.

Section 4.2 examines the two data sets studied in this work. The first is the real sensor data obtained from the RAS mentionned in Section 4.1, supplemented by operator-entered data during routine daily inspections of the facility. This dataset serves to show the viability of the aMSCRED model to be used in the context and timeframe of aquaculture systems. The second dataset is a synthetic dataset, generated with parameters which characterize multivariate timeseries data similar to the RAS dataset.

The next section, Section 4.3, elaborates upon the steps taken to generate signature matrices from the input timeseries data, and to put them into the format the aMSCRED model expects as input.

Section 4.4 outlines the process used to tune the aMSCRED model hyperparameters, as well as presents a complete list of the hyperparameter values tested in this work.

The training, validation, and testing process are detailed in Section 4.5, alongside other implementation details relevant to this work.

The methodology employed to detect anomalies from the trained aMSCRED model is described in Section 4.6, establishing the concept of the anomaly score and how it is used to predict whether or not a given input is anomalous. This is followed by an explanation of the evaluation metrics used to measure a trained model's anomaly detection performance, in Section 4.7.

This chapter wraps up with a section on root cause identification, Section 4.8, which explains the process and metrics used to measure the model's capacity to identify which timeseries are responsible for any given anomaly. It is followed by a short section on the methods used to measure the model's robustness to noise in input data, in Section 4.9.

4.1 The Facility

This research was conducted, in partnership with the University of Manitoba and the Myera Group Campus in Manitoba, Canada. This facility

works closely with agricultural producers, government, and academia to create innovative high-tech solutions in the agricultural-technology space. The primary activity at this location is running a pilot fish rearing and Arctic Char hatchery and developing the technology in preparation for scale-up to a full-size production facility in the near future. The end-goal is to create a sustainable fish farming industry in Manitoba in which the waste produced by aquaculture is used to produce secondary products, reducing the waste footprint of the operation as much as possible.

The research and development at this facility focuses on the development of an innovative tank design to improve energy efficiency in terms of water-transport, development of a distributed sensor system to closely monitor critical system components and parameters, facility automation for consistency and traceability of actions (e.g. when feeding, when were they fed and how much feed was distributed), and physiological experiments to learn how to fully understand the biology of the fish species being farmed. For example, an exercise program where water flow-rate is increased for a portion of the day, encouraging the fish to swim faster and exercise. Another, experimenting with different feed types, specifically creating a range of plant-based formulations. Yet another, experimenting with day-night cycles to mimick the natural northern environment from which the species originate.

4.1.1 The Tank Design

The system in question belongs to the category of a Recirculating Aquaculture System (RAS), a closed-loop aquaculture system in which water is cleaned, filtered, and recirculated through the system continously. This type of system has the advantages of both reducing costs and reducing environmental impacts with regards to water usage and waste discharge.

The system is seperated into five principal components, as shown in Figure 4.1.

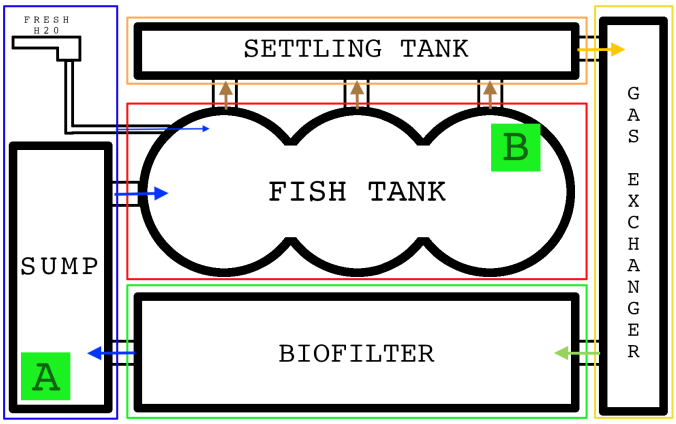

Figure 4.1: Top-view of the principal components of the recirculating aquaculture system (RAS). Recirculated water and fresh make-up water are pumped from the sump into the fish tank, where it acquires waste products from the fish. The solid waste is filtered out in the settling tank. Liquid waste is converted in the biofilter, from ammonium to nitrates. The gas exchanger columns allow dissolved CO_2 to degas while simultaneously replenishing DO. The cleaned water flows back to the sump, where it is recirculated. The location of each sensor pack within the system is labelled by A and B, respectively.

The fish tank is the largest portion of the system volume and is where the fish reside. It is here that fresh water is initially pumped into the system

and where the filtered recirculated water is pumped back. The tanks consist of connected circular cells that can be left open or blocked with a grating in order to segregate different groups of fish. A top-view of a single cell of the tank is shown in Figure 4.2. The water current in each cell is circular, with most fish swimming against the current. Figure 4.3 shows a side-view of the tank (on the left) with the settling tanks shown on the right. The grey box shown on top the fish tank is sensor pack B.

Figure 4.2: Top-view of a single circular cell from the fish tank. The water flows in a circular pattern, with the majority of the fish swimming against current at a leasurely pace to stay in place.

Throughout the day, the fish produce excrement (both solid and liquid waste) and also consume oxygen (O_2) while producing CO_2. These changes in the water renders the water harmful to the fish physiology, thus the excrements need to be filtered-out, the CO_2 must be stripped, and dissolved O_2 replenished.

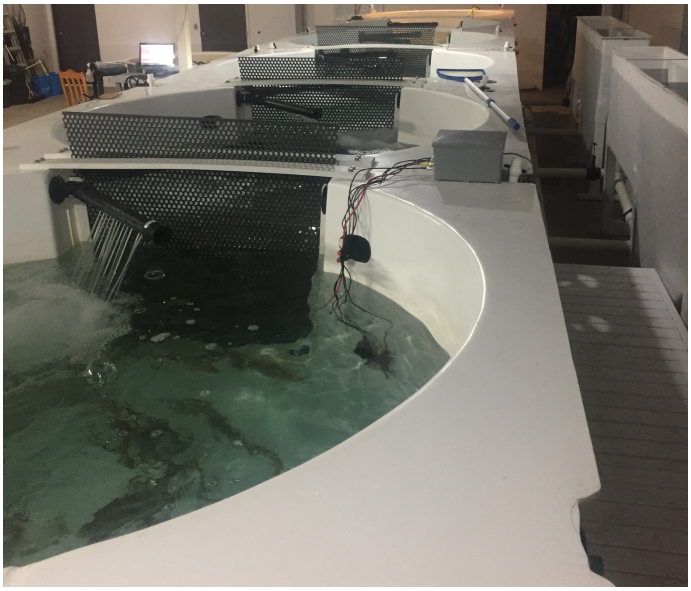

Figure 4.3: Side-view of the fish tank (left of walkway). Sensor pack B is seen in this photo, with its probes submerged in the tank. To the right of the walkway, the settling tanks.

Water flows into the settling tanks, where solid waste settles to the bottom of the tank and the remaining water is skimmed off the surface and flows into the next stage, the gas exchange columns. A flushing mechanism clears out settled solid waste routinely.

In the gas exchange columns, CO_2 is stripped from the water and O_2 is dissolved into the water. This is achieved by creating a mist of water droplets, ensuring a high-surface area to volume ratio with the surrounding air, facilitation diffusion of dissolved gases. The replenished water then flows through the biofilters, shown in Figure 4.4, where harmful ammonia and ammonium are converted to less harmful nitrates by nitrifying bacteria. The bacteria reside on growth media which consist of plastic beads with a

high surface-area to volume ratio. The growth media is constantly mixed and agitated by a submerged air-pump to prevent anerobic conditions from forming in the biofilter. Finally, the cleaned and filtered water flows into the sump where it is pumped back into the fish rearing tank, in order to repeat the whole process.

Figure 4.4: RAS biofilter is filled with growth media on which nitrifying bacteria reside. They convert ammonia and ammonium in the water into nitrates, which are safer for the fish. The bubbles in the middle of the tank are from a submersed air-pump which ensures there is enough oxygen supply available to the bacteria.

Two sensor packs with identical sensors are placed in this system in such a way to capture the gradient of the water quality across both points in the

system. In the future, sensor packs will be outfit to each component of the tank. However, in this pilot-scale tank it was not deemed necessary nor cost-effective. Sensor pack A is placed in the sump, where the water has been filtered and is ready to be recirculated (at its cleanest), and sensor pack B is placed in the fish tank, where the water is at its dirtiest. The sensors used in these sensor packs are detailed further in the next section.

A feature of the RAS system is the ability to control water-flow in such a way as to alternate between high-flow and low-flow states throughout the day and throughout the lifecycle. This has dual benefits. The first are health benefits to the fish. Putting them through cycles of alternating exercise and rest improves meat quality, while also reducing aggressive behaviour amongst them. The second benefit is reducing energy costs during low-flow states, as less pumps need to be powered-on.

The challenges that arise in this type of dynamic system are that transitions between velocity states can rapidly change system parameters. For example, during ramp-up to a high-velocity state, a sudden drop in DO concentrations could occur due to the fish entering into a state of increased oxygen consumption.

4.1.2 The Sensor System

The sensor system installed in the RAS system described in Section 4.1.1, consists of two seperate sensor packs with identical loadout of sensors. Each sensor pack contains a Raspberry Pi running Raspbian and custom software

to read and relay sensor data to the central data monitoring server. Each sensor pack has five submersible probes for measuring the factors described in Section 2.3: temperature (RTD), dissolved oxygen (DO), pH (PH), electrical conductivity (EC), and oxygen reduction potential (ORP), from Atlas Scientific's line of EZO sensor products. The probes are interfaced via a USB interface to the Raspberry Pi. The electronics of each unit are housed in a waterproof container and powered via power-over-ethernet (PoE). More details on the construction of the sensor packs are discussed in Appendix A. A picture of Sensor Pack A is shown in Figure 4.5, and it's innards are shown in Figure 4.6.

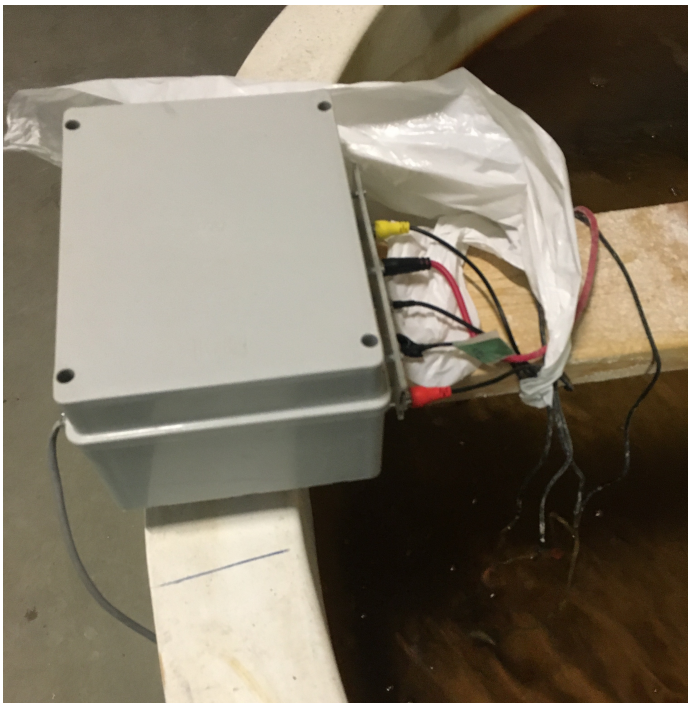

Figure 4.5: A sensor pack sits on the side of the sump with 5 submersible probes measuring DO, pH, temperature, ORP, and EC at regular intervals.

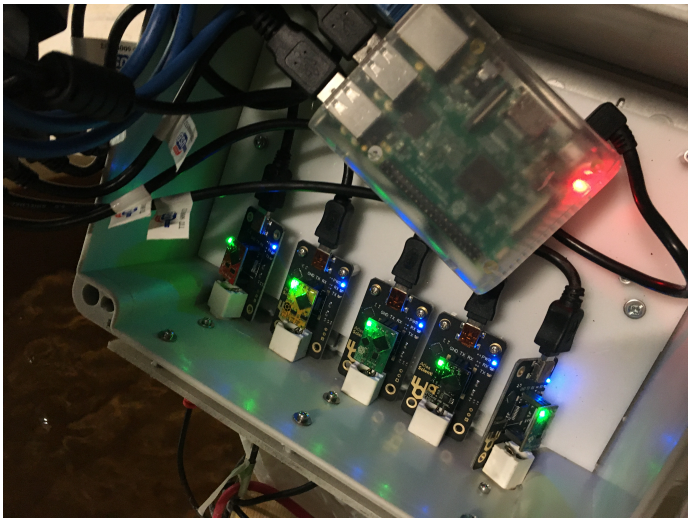

Figure 4.6: Inside the sensor pack, the Raspberry Pi is shown, as well as the 5 EZO sensor boards interfaced via a USB carrier board.

Each sensor pack is placed at two opposite ends of the closed-loop RAS system in order to gauge the gradient of each measured variable across the system. Due to budgetary reasons only two sensors packs are setup at the facility at this time, but in the future this could grow to as much as one unit per tank cell. Thus, the anomaly detection system must be flexible when needing to add additional timeseries.

The sensors are sampled at 30 second intervals with measurements stored on a central server via a customized monitoring system. Thresholds for normal operation of each timeseries can be defined, such that whenever values exceed these thresholds, the technicians at the facility are alerted and corrective action can occur. Further details regarding specifics of the monitoring system are presented in Appendix A.

4.1.3 Additional Sources of Data

Technicians at the facility make daily reports via data-entry on a tablet, regarding both some qualitative aspects and measurements that are not yet integrated into the electronic sensor system either for lack of availability of electronic sensors or lack of funds to purchase expensive detectors.

The qualitative aspects include which tanks/cells were fed as well as the responsiveness of the fish to being fed (were the fish jumping for joy, or unreactive). Using widgets on the tablet, such as a sliding-scale to indicate responsiveness to feed, this qualitative "gut-feeling" can be translated into quantitative numerical data.

Other recorded values include dissolved concentration of ammonia, nitrate, and nitrite, from water chemistry tests that must be done manually via titration or some form of colorimetry. Automated systems which can carry-out these measurements are far too expensive to be able to scale and thus are carried out manually at this stage of the project.

Finally, the number of mortalities and jumpers (fish having jumped out of the tanks and not necessarily dead) are recorded as well. Since these occurrences are quite rare, they are used to label data points as anomalous for later use when measuring anomaly detection efficiency of different trained models. This process will be elaborated upon further in section 4.2.1.

4.1.4 Limitation of the Monitoring System

The monitoring system put in place at this facility prior to this work utilizes thresholds to determine when sensor values are within normal ranges. This current implementation has some issues that need to be addressed; in some instances, complex issues have arisen that still fell inside the normal operating thresholds when looking at each timeseries individually, but when considered together, the issue could have been detected. In other events, certain sensors started to fail and yielded erroneous readings, experienced calibration drift, or fail altogether and give false readings that appear to remain in the normal region of operation. Another foreseeable drawback is a breakdown in reliability that occurs when transitioning from the low-flow to high-flow states, and vice-versa. Furthermore, there are other challenges with regards to the threshold-based system not being able to detect when sensors are failing in the scenario where they might be returning false readings that are within the acceptable thresholds and thus do not trigger an anomaly detection event. Characterization of normal system state may not be enough, there is a requirement to characterize a family/class of 'healthy' states that can be used across different modes of operation, i.e. some parallels can potentially be learned between healthy conditions in the low-flow and high-flow states, or between tanks with younger cohorts versus tanks with older cohorts.

Therefore, the need to augment the existing system with an anomaly detection system that can learn what the normal state of the system looks

like across all sensors and across time becomes necessary to reduce needless losses that can occur during anomalous events.

The monitoring system currently in place requires an additional layer that can learn what are normal states, as well as what are normal transitions between states, in terms of multivariate variations. The ability to learn this unsupervised allows the monitoring system to detect anomalies that are not forseable (e.g. unknown chemical contamination) or detectable by the existing threshold based system. The system must also be adaptive and be able to continuously relearn what a normal state looks like in the RAS, as this tends to evolve over time as the cohorts of fish grow larger and their feeding and general behaviour changes as well, carrying with it varying consequences on water quality.

4.2 Considered Datasets

Two datasets are used to train and subsequently evaluate the model proposed in this work; the first one is electronic sensor data obtained from the RAS presented in Section 4.1, supplemented by routine manual measurements. This dataset serves to show the viability of the examined models to be used in the context and timescale of a RAS.

The second dataset is synthetic multivariate timeseries, generated with parameters which characterize data similar to the data found in the RAS system (similar periodicities and correlation). The purpose of using synthetic

datasets is being able to measure the effect of modifying certain properties of the data on the outcome of the model's performance during testing. In particular for this work, the synthetic dataset has a controllable noise factor. The amplitude of the noise can be increased or decreased via a noise coefficient and added to noiseless synthetic data. As such, identical datasets differing only in random noise level, ranging from 0% up to 30%, can be generated in order to test the model's robustness to noise in the dataset.

4.2.1 RAS Dataset

The dataset used in this work consists of measurements from the ten electronic sensors described in 4.1.2, recorded between the period of May 9th, 2018 to June 29th, 2018. This period was chosen because it follows the installation of the 2nd sensor pack being added to the system, and ends prior to a PH-B probe breaking on June 30th, 2018. The fish cohort in the RAS during this time were between 18 and 24 months old. During this period, the sensors were sampled asynchronously every 5 seconds. Thus, readings are resampled into 1 minute bins, yielding a final dataset on the order of 10^5 timesteps.

The sensors in this dataset are labelled by their sensor type and then A or B, if it is in sensor pack A or B respectively. (e.g. the dissolved oxygen probe at sensor pack A is labelled DO-A).

The manually entered operator data described in Section 4.1.3 were entered once daily. Therefore, values were linearly interpolated to get values

for each 1-minute bin, in order to line up with the electronic sensor data. Each timeseries in the dataset is subsequently normalized, using min-max scaler. Furthermore, the data on jumpers and mortalities are used to label datapoints as normal or anomalous (0 or 1 respectively). Days where the jumper count or anomaly count is greater than zero get marked as anomalous, setting the true class column to 1 for every timestep 12 hours prior to and 12 hours following the time the operator entered the data. This true class label is leveraged only during the testing phase to measure the anomaly detection performance of the model. It is not used during training, as the model proposed in this work learns in an unsupervised manner.

The correlation matrix of this dataset over all timesteps, shown in Figure 4.7, demonstrates the overall pairwise correlation coefficients between each timeseries in the dataset. The pairwise correlation coefficients are the basis on which the model in this work learns the state of the system, by observing changes in correlation coefficients.

For example, timeseries 2 and 3 (water temperature at sensor pack A and at sensor pack B, respectively) each have a correlation coefficient of approximately 0.9 with timeseries 13 (incoming fresh water temperature). This is expected in this RAS, since fresh water is pumped directly into fish tank, thus affecting its temperature as well. An example of an anti-correlated pair is timeseries 11 and 12, which are nitrate and nitrite respectively, with a correlation coefficient around -0.6. This is expected as well, since nitrites convert to nitrates in the RAS system's biofilters. For every nitrate molecule

created, nitrite is consummed.

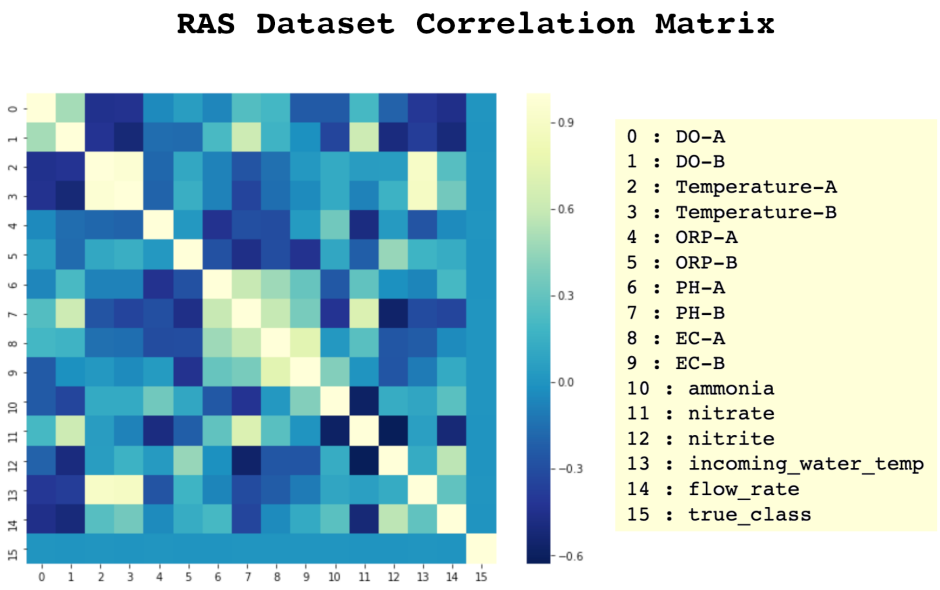

Figure 4.7: Correlation matrix of the RAS dataset, taken over the entire dataset (prior to any further anomaly injection).

The Real Anomalous Event

At timestep $t = 45,628$ to $t = 45,689$ there was an issue that caused several fish deaths. The post-mortem analysis revealed rapid fluctuations in ORP and EC levels in the growout tank. Rapid fluctuations in the nitrate level were suspected to also be an issue since there were large variations in nitrate levels days prior and after the mortalities. The issue is nitrate only gets measured once daily, thus the actual fluctuation was not captured in the monitored data. Thus, an anomaly is injected into the nitrates reading

(following procedure outlined in Section 4.2.4) to emulate this behaviour. However, the corresponding EC and ORP values are left unmodified.

Dummy Timeseries for Error Reduction

During initial training of the model during development, it showed that there was a lot of reconstruction error on the last column of the signature matrices, invariant of the ordering of the timeseries. This is most likely due to some issue with the implementation of the padding in the convolutional layers. However, a simple fix to mitigate this effect was to add a dummy timeseries set to zero, which effectively removed the high degree of error that tended to accumulate on the last timeseries of the signature matrices.

4.2.2 Synthetic Dataset

The second dataset is synthetically generated multivariate timeseries. It consists of $n = 32$ timeseries and $T = 200,000$ timesteps. Each individual timeseries, x_i, is composed of a randomly selected periodic function, $f(\bullet)$, for $t : t \in [0, T)$

$$x_i = f(w * t + t_0) \tag{4.1}$$

where $f(\bullet)$ is a random choice of $\sin(\bullet)$ or $\cos(\bullet)$ in this case, with a period, w, such that $w \in [40, 50)$, and phase shift $t_0 \in [50, 100)$.

Periodic functions were chosen with a tight range on periodicity and randomly chosen between cos() and sin() in order to obtain a range of timeseries

that are mostly correlated or anti-correlated, as the RAS dataset showed similar correlation behaviour.

In order to reproduce the effect of noise present in real-world data, a noise parameter is added onto the synthetic timeseries signal, scaled by a tuneable noise coefficient, λ. Equation 4.1 becomes

$$x_i = f(w * t + t_0) + \lambda * \mathcal{N}(0, 1) \tag{4.2}$$

where $\mathcal{N}(0, 1)$ is a vector of length T with values randomly selected from the gaussian distribution.

4.2.3 Anomaly Types

The RAS system and the sensor system presented above can undergo a multitude of classes of events that are considered anomalous but might manifest different signatures in the dataset. For example, a failing sensor (be it membrane wear or faulty electrical connection) may create a burst of erratic readings with no discernable pattern.

On the other hand, other anomaly types can introduce systematic changes to the dataset. The buildup of a biofilm in the RAS system has been observed that can impact sensor readings, especially for pH, EC, and DO, which depend on surface area and good flow of the water across the sensor probe. If probes are cleaned regularly, this biofilm does not lead to a noticeable impact on sensor data. However, if left long enough, there is systematic error intro-

duced which will add some constant bias/drift to the sensor readings. This type of anomaly increases in severity continuously and linearly with time on the affected sensors until a cleaning event, at which time a discontinuous jump occurs and the next sensor measurement is back at the true value.

The remaining class of anomalous events, and the most important for the reduction of mortalities in the RAS, are anomalies in the system parameters (i.e. water chemistry) that could have adverse effects on fish health. These manifest as some correlated changes between subsets of the multivariate timeseries. The dataset from the RAS does not yet have enough instances of this type of anomalous event to carry-out statistical analysis to determine signatures left behind by different issues. While this can be done in theory, the principal issue is that in order to gather the data required to carry out such an analysis, it is necessary for mass-mortality events to occur in order to gather necessary data, which is undesirable from the business perspective.

The hope is that the aMSCRED model will be able to learn and detect these signatures in this system, bypassing the need to learn of system issues after the fact and reducing overall mortality rates in the RAS.

4.2.4 Anomaly Injection

The RAS dataset only contains one instance of an anomalous event and the synthetic dataset by nature does not contain any anomalies after being generated. For this reason, artificial anomalies are injected into both datasets in order to increase the total number of anomalous events to 5. Five is

chosen because it increases the resolution of the measurement of the anomaly detection performance without increasing the ratio of anomalous to normal data in the testing data by a significant amount (roughly 0.3%). This also corresponds to the number of anomalies found in the datasets used by Zhang et al., to serve as a basis for direct comparison.[1] The anomalies are only injected into the testing portion of the dataset (i.e. the last 40% of the timesteps).

Different classes of anomalies might leave different signatures in how they modify a timeseries. Without any reference material in the literature and a lack of anomalous events in the RAS dataset to analyse, no assumptions are made in this work as to how these anomalies should be characterized in the context of a RAS. The most trivial anomaly type will be used, which is simply random noise without any underlying signal. Injected anomalies are generated as a vector of random noise from the uniform distribution (in the interval $[0, 1)$). They overwrite the original values of the target (normalized) dataset for the timeseries and over the timesteps that should be detected as anomalous.

4.2.5 Anomaly Labelling

An additional column is appended to the datasets, which labels each timestep as either normal (0) or anomalous (1).

In the case of the RAS dataset, the class labels are obtained from operator data collected on a daily basis during their rounds of the facility. Whenever

mortalities are observed, or fish behave oddly (erratic movement, not feeding, agressive behaviour, etc...) the timesteps for that day are labelled as anomalous. Every other timestep is labelled as normal by default.

In the case of the synthetic data, every timestep is initially assigned the normal label (i.e. 0) and subsequently changed to anomalous (i.e. 1) whenever an anomaly is injected at a given timestep.

The class labels are excluded during the model training and model validation, as this is an unsupervised learning model. The class labels are only used in the testing phase to measure anomaly detection performance of the trained model. Knowledge of the true class of each datapoint is necessary to measure Precision, Recall, and F1 scores, in terms of determining counts of true positives, false positives, and false negatives.

4.3 Data Preprocessing

Each dataset undergoes initial preprocessing that consists of generating a set of signature matrices for each sampled timestep (i.e. every u timesteps, where u is the gap size between subsequent windows) and assembled into groups of h sequential signature matrices (where h is the history size hyperparameter of the recurrent component). A group of h signature matrices serves as a single input to the model, yielding the input tensor $\mathcal{X}^{t,0} \in \mathbb{R}^{h \times n \times n \times s}$ for the group of subsequent signature matrices ending at time t.

4.3.1 Signature Matrix Labelling

Signature matrices are generated as defined in Section 3.2.2. In this implementation, the number of channels (i.e. number of windows) and respective window sizes are left as tunable hyperparameters. The totality of tested hyperparameter values are listed in Section 4.4.

The anomaly class labels from the dataset must be transferred to the signature matrices. A signature matrix represents a range of timesteps, but the anomaly labelling from the original dataset is defined on a per timestep basis. Signature matrices obtained from a window in the original dataset containing at least one anomalous timestep are labelled anomalous. Conversely, if the window contains no anomalous data, then it is labelled normal. Formally, this process labels signature matrix M^t via the labelling function, l,

$$l(M^t) = \begin{cases} 0, & \text{if } l(x^{t-w_{\min}:t}) = 0 \\ 1, & \text{otherwise.} \end{cases} \tag{4.3}$$

where $x^{t-w_{\min}:t}$ is the timeseries segment X^w between timestep $t - w_{\min}$ and timestep t, and $w_{\min}$ is the smallest window size.

4.3.2 Model Input Preparation

The model expects inputs of dimension $h \times n \times n \times s$, where h is the history size (number of sequential signature matrices per input), n is the number of timeseries, and s is the number of channels (or scales) for which

the signature matrix is generated.

In other words, the inputs are expected to be sequences of h consecutive signature matrices, sampled from timestep $t - h$ to t.

Inputs are shuffled prior to each training step in order to prevent overfitting as an effect of having all input data provided in the same sequence for every training run.

4.4 Hyperparameter Tuning

Hyperparameter tuning adaptively selects optimal structural parameters for the the model, parameters that are not learned during training. This is accomplished by training models multiple times for each set of hyperparameters and comparing the anomaly detection results obtained for each model. Table 4.1 identifies the hyperparameters of aMSCRED, as well as the values tested during the hyperparameter tuning process. They are broken out into three categories. First, the signature matrix generation hyperparameters. Second are recurrent component hyperparameters and third are the convolutional component hyperparameters.

In total, there are roughly 5000 hyperparameter permutations to test in the search space. This is accomplished using a basic grid search, where every possible permutation of hyperparameter values given in Table 4.1 are used to train, validate, and test the model. A set of anomaly detection performance metrics, presented later in Section 4.7, are tracked for every point in the

Table 4.1: Model hyperparameters and values tested during hyperparameter tuning.

HYPERPARAMETER	**Values**
Window Sizes	$[10, 30, 60]$,
	$[5, 15, 30]$,
	$[5, 30, 90]$
Window Step	$1, 5, 10$
Corr. Function	mscred,
	pearson,
	fractional time,
	exponential time
History Size	$3, 5, 7, 9$
Attention Layer	none,
	mscred
# Conv. Kernels	$[16, 32, 64, 128]$,
	$[32, 64, 128, 256]$,
	$[64, 128, 256, 512]$
Conv. Kernel Size	$[3 \times 3, 3 \times 3, 3 \times 3, 3 \times 3]$,
	$[3 \times 3, 3 \times 3, 2 \times 2, 2 \times 2]$,
	$[2 \times 2, 2 \times 2, 2 \times 2, 2 \times 2]$
Conv. Strides	$[1 \times 1, 2 \times 2, 2 \times 2, 2 \times 2]$,
	$[2 \times 2, 2 \times 2, 2 \times 2, 2 \times 2]$

search space and later used to determine which set of hyperparameters yields the best-suited model.

The values present in this table represent a small subset of the countless possible values and permutations thereof. However, in the interest of reducing total training times, the grid search was limited to this limited scope. This subset starts from hyperparameters in the MSCRED implementation and attempts values in a small neighbourhood around them. Future work to implement a simple random search with a broader hyperparameter search

space, especially for the convolutional component parameters, would search a larger volume of the total search-space and potentially yield superior models in terms of their anomaly detection performance. More will be said on this in Section 5.6.

A breakdown of the hyperparameters in the different components of the model are presented in the remainder of this section.

4.4.1 Signature Matrix Hyperparameters

The principal parameters in signature matrix generation are the choice of window size, gap size between windows, and the correlation function.

The notation given for window size in Table 4.1 denotes a list of window sizes used for each channel. For example, $[10, 30, 60]$ denotes signature matrices containing three channels, sampled with windows of size 10, 30, and 60 timesteps, respectively.

The window step parameter, also called the gap size, denotes choices for gap size between the sampling windows of their respective signature matrices.

The correlation function hyperparameter selects between the four proposed correlation functions, elaborated upon in Section 3.2.2.

4.4.2 Recurrent Component Hyperparameters

History size hyperparameter refers to both the number of sequential signature matrices in a given input, as well as the number of recurrent steps in

the recurrent component of the model.

The Attention Layer hyperparameter refers to the choice of attention layer placed at the output of the recurrent layers; *none* bypasses the attention layer by feeding ConvLSTM outputs to the convolutional decoder directly, *mscred* refers to the scaled dot-product attention layer implemented in paper [1].

4.4.3 Convolutional Component Hyperparameters

The last three hyperparameters relate to both the convolutional encoder and convolutional decoder. For example, the choice of number of kernels $[16, 32, 64, 128]$, kernel sizes $[3 \times 3, 3 \times 3, 2 \times 2, 2 \times 2]$, and strides $[1 \times 1, 2 \times 2, 2 \times 2, 2 \times 2]$ means there are 4 convolutional layers in total. The first layer has 16 kernels of size 3×3 sampled in strides of 2×2. The second layer has 32 kernels of size 3×3 sampled in strides of 2×2, and so on. The convolutional decoder uses identical parameters to the convolutional encoder on any given training run.

4.5 Training, Validation, and Testing

The dataset is split into training data and validation data, X_{train} and X_{valid} respectively (the first 60% of datapoints), and training data, X_{test} (the remaining 40% of datapoints). The training data and validation data contain only datapoints from the normal class. That is, only the testing data portion contains anomalous data. For the RAS dataset, datapoints in the

normal class are the most abundant, with anomalous data representing only a fraction of total datapoints ($< 1\%$).

Using backpropagation and the ADAM optimizer, models are trained for for N epochs. At the end of each epoch, the validation data is used to measure the model's loss for this epoch. After all N epochs, the model with the smallest reconstruction loss is tagged as the best epoch.

The model from the best epoch is subsequently tested with the testing data to measure its anomaly detection performance. The performance is measured in terms of Precision (Pre), Recall (Rec), and the F1 Score (F1), which are formally defined in Section 4.7.

During the training phase, the model essentially learns what normal states are for the system. This is accomplished by optimizing (minimizing) reconstruction loss. In other words, given some input data, how accurately will the model will be able to reconstruct the input. Since all training data is normal data, the objective is to reduce reconstruction loss as much as possible, i.e. the reconstructed input is as similar as possible to the original input.

When abnormal data is input into a trained model, the reconstruction error should be greater than some expected threshold. This threshold is obtained using the testing and validation results for this model. This phenomenon is exploited as the mechanism to detect anomalies during the testing phase.

During testing, the trained model is fed a mixture of both normal and abnormal data. By comparing the reconstruction error given the input data,

the model predicts if the input data is normal or anomalous. This anomaly detection process is elaborated upon further in Section 4.6.

4.5.1 Implementation Details

The aMSCRED model is implemented in Keras and TensorFlow 2.0. Hyperparameters are tuned using TensorFlow's HParams plugin. TensorFlow's graphical interface, Tensorboard, is configured to compare metrics obtained for different sets of hyperparameters, one for each run.

The sets of models were trained using the tensorflow-gpu docker container on an Intel server with 64 GB RAM and 4 Nvidia 1080Ti GPUs. In order to fully utilize the parallel compute afforded by these GPUs, the Keras multi-gpu model interface is used to split the model training batch-wise across 4 GPUs simultaneously, only using the CPU for merging batches after the parallelizable computations are completed.

Total runtime for the entire search space of hyperparameters took approximately 320 hours, averaging 90 seconds per epoch per set of hyperparameters.

4.5.2 Training Loss

Loss during the training phase, also refered to as the reconstruction error, is computed using Equation 3.24 in Section 3.2.6. Reconstruction error is measured and recorded for every training epoch. It is not directly useful

for the anomaly detection process, rather it is only used during training to optimize the weight and biases of the model.

4.6 Anomaly Detection Methodology

This section details how the anomaly score, $s_\theta(R^t)$, is determined given the residual matrix, R^t, as well as how anomalies are predicted given the anomaly score.

4.6.1 Residual Matrix

For any given input signature matrix, M^t, at time step t, and its corresponding reconstructed matrix $\hat{M}^t$, the residual matrix, R^t, is given by

$$R^t = M^t - \hat{M}^t. \tag{4.4}$$

A simplified example of this process is illustrated in Figure 4.8, showing only a single channel of the signature matrix. The residual matrix is equivalent to the element-wise reconstruction loss when comparing the input signature matrix to the reconstructed matrix.

4.6.2 Anomaly Score, s_θ

The anomaly score, s_θ, of a residual matrix, R^t, is defined as the number of poorly reconstructed pairs by the model, where the element-wise error

Figure 4.8: Residual matrix is the difference between input signature matrix and its corresponding reconstructed matrix.

in poorly reconstructed pairs is greater than some threshold, θ. In other words, it is the number of elements $r_{ij}^t \in R^t$ such that $|r_{ij}^t| > \theta$, where θ is an empirically determined threshold that maximizes the likelihood of later correctly classifying a residual matrix as normal or as anomalous. More formally, this is given by

$$s_\theta(R^t) = |\{r_{ij}^t \in R^t \mid r_{ij}^t > \theta\}|. \tag{4.5}$$

The anomaly score is a monotonically-decreasing step function, with range from n^2, when $\theta = 0$, to 0, when $\theta = \max\{R^t\}$, where $\max\{R^t\}$ is

the largest element-wise reconstruction loss value in R^t. An example of what this function looks like is shown in Figure 4.9.

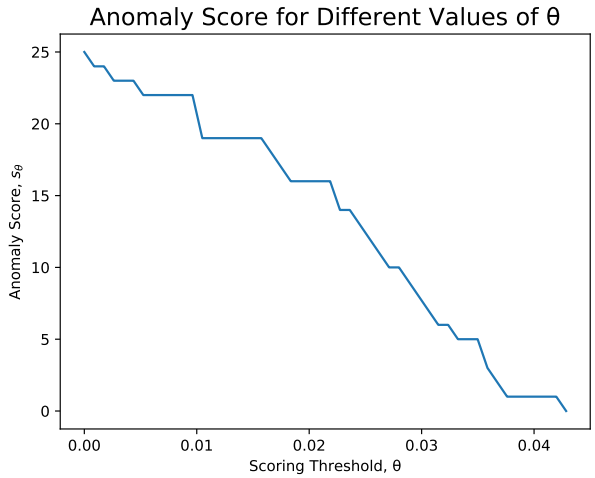

Figure 4.9: The anomaly score as a function of θ is monotonically decreasing step function.

The value for θ is found dynamically for any given dataset, by empirically deducing a value of θ. This places an upper bound on anomaly scores for all residual matrices that are known to the system and are known to be normal (i.e. training and validation data). The upper bound in this implementation is set to one more than the square root of the total number of timeseries in the dataset, $\sqrt{n} + 1$. In the case of the RAS dataset, this is $\sqrt{16} + 1 = 5$.

During the testing phase, the anomaly score is computed for each residual matrix. A depiction of this process is shown in Figure 4.10, using the residual matrix from Figure 4.8.

Figure 4.10: The anomaly score of a residual matrix is the number of elements in said matrix which lies outside the threshold, θ. The offending elements in this figure are highlighted in red.

4.6.3 Anomaly Classification and τ

The anomaly threshold, τ, is used to classify residual matrices as normal or anomalous depending whether its anomaly score is lesser than or greater than τ, respectively. Formally put, the predicted classification of each residual matrix during testing phase, y_{pred}, is defined as

$$y_{\text{pred}} = \begin{cases} 0, & s_\theta(R^t) \leq \tau \\ 1, & s_\theta(R^t) > \tau \end{cases} \tag{4.6}$$

where $s_\theta(R^t)$ is the anomaly score of the residual matrix obtained at timestep t, for a fixed value of θ (shared across all signature matrices), and 0 and 1 represent normal and anomaly, respectively.

The anomaly threshold, τ, is determined after the training phase, by finding the maximal anomaly score obtained during validation and scaling it by some coefficient, β. Specifically, τ is defined as

$$\tau = \beta \max s_\theta(t_{\text{valid}}), \tag{4.7}$$

where $s_\theta(t_{\text{valid}})$ are the anomaly scores over the validation period, and $\beta \in [1, 2)$ is chosen such that it maximizes the F1 score of the validation period. This process is illustrated in Figure 4.11.

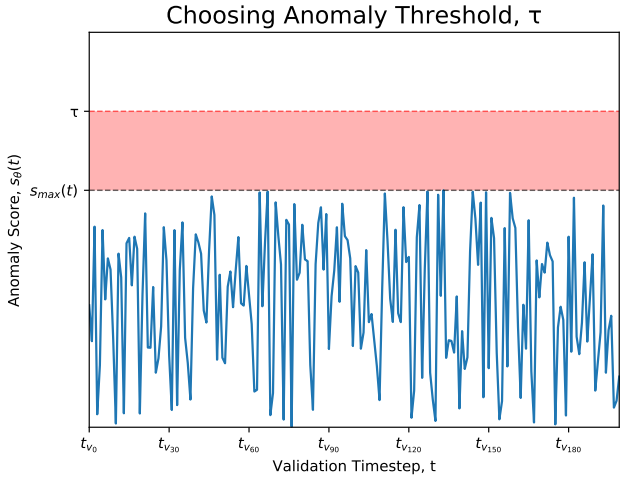

Figure 4.11: The anomaly threshold, τ, is chosen based on the maximum anomaly score obtained during the validation period, denoted as s_{max} in this figure. The red zone represents the buffer area afforded by the chosen scaling coefficient, β

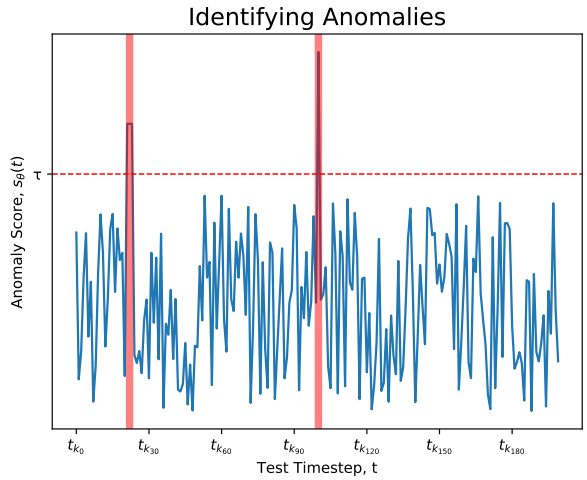

Figure 4.12: During the testing phase, timesteps for which $s_\theta(t_{\text{test}}) > \tau$ are predicted as anomalies. In this figure, these timesteps are highlighted in vertical red bars.

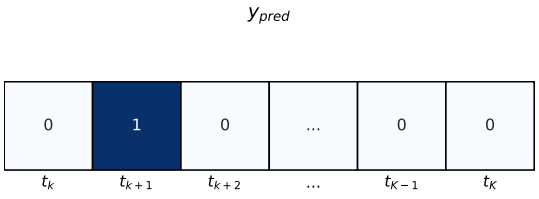

Figure 4.13: y_{pred} is a vector containing classification labels for each residual matrix obtained during the testing phase. Data classified as normal is labelled 0, whereas data classified as anomalous is labelled 1.

During testing, τ is used to predict all timesteps which are anomalous, as shown in Figure 4.12. Every anomalous timestep t_k is labelled as such and stored in vector $y_{\text{pred}}(t_k)$, as depicted in Figure 4.13.

4.7 Evaluation Metrics

Anomaly detection of timeseries data is the process of detecting data points in the set that do not fall within some predictable range, with respect to the rest of the data contained. In order to evaluate the anomaly detection performance of a trained model, the predicted anomalies and true anomalies must be compared and measured via some collection of metrics. Standard metrics for evaluating anomaly detection performance are Recall (Rec), Precision (Pre), and F1 score (F1). They are formally defined in this section.

These metrics involve computing ratios between true positive counts (*TP*, anomalous data predicted as an anomaly), true negative counts (*TN*, normal data predicted normal data), false positive counts (*FP*, normal data predicted as an anomaly), and false negative counts (*FN*, anomalous data predicted as normal). These counts are obtained by comparing entries in the $\boldsymbol{y}_{\text{pred}}(t_k)$ and $\boldsymbol{y}_{\text{true}}(t_k)$ vectors. For example, when $\boldsymbol{y}_{\text{pred}}(i) = \boldsymbol{y}_{\text{true}}(i) = 1$, for some $i \in t_k$, this is a TP. When $\boldsymbol{y}_{\text{pred}}(i) = 0$ and $\boldsymbol{y}_{\text{true}}(\boldsymbol{i}) = 1$, this is a FN. And so on.

4.7.1 Precision Score

Precision, Pre, is defined as the ratio of true positives to all positive predictions,

$$\text{Pre} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.8}$$

where TP is the number of true positives obtained, FP is the number of false positives obtained, and Pre $\in [0, 1]$.

In essence, Pre measures the likelihood that a timestep predicted as an anomaly is a true positive. Pre is dependent on the anomaly threshold, $\tau \in [0, n^2]$, since the ability to obtain either a TP or a FP necessitates τ to be smaller than the anomaly score for at least one residual matrix. Figure 4.14 depicts this dependency, using some example data obtained during an arbitrary testing phase. The ideal choice of τ, when referring to this figure, is to pick the smallest τ such that Precision is still relatively high, near $\tau = 8$.

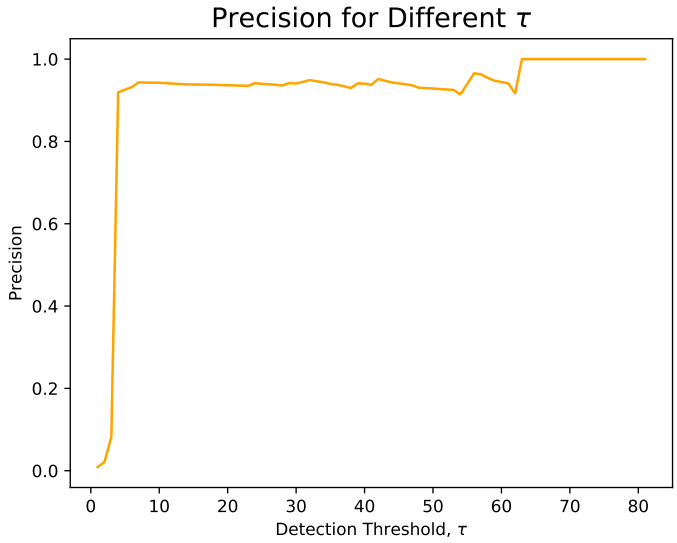

Figure 4.14: Precision obtained when using values of τ from 0 to n^2.

In the limiting case, when $\tau = 0$, all timesteps are predicted as anomalies. Thus, Pre will attain a local minimum. On the other limit, when $\tau = n^2$ (where n is the number of timeseries), all residual matrices will score less

than or equal to n^2, thus no timesteps are predicted anomalous. Thus, Pre in this case is undefined due to division by 0. In this circumstance, the value of Pre is set to 0 as a proxy, since this is an undesirable result, warranting the worst score.

4.7.2 Recall Score

Recall (Rec) is defined as the ratio of true positives to total number of samples that should have been predicted as anomalies. In essence, it describes the likelihood of all anomalous events being detected by the model. It is given by

$$\text{Rec} = \frac{TP}{TP + FN} \tag{4.9}$$

where Rec $\in [0, 1]$.

Recall starts at 1 when $\tau = 0$, since every timestep is predicted as anomalous (positive). Thus, in this case, FN = 0 and Rec simplifies to Rec $= \frac{TP}{TP} = 1$. As τ approaches n^2, Rec drops off to zero, as all results are predicted negative (i.e. normal), thus Rec $= \frac{0}{0+FN} = 0$. This relationship between choice of τ and Rec is depicted in Figure 4.15.

4.7.3 F_1 Score

While it is desirable to maximize both Recall and Precision, there is a trade-off between both values as shown in the previous sections, which is dependent on the threshold value, τ. Smaller values of τ tend to favour Rec,

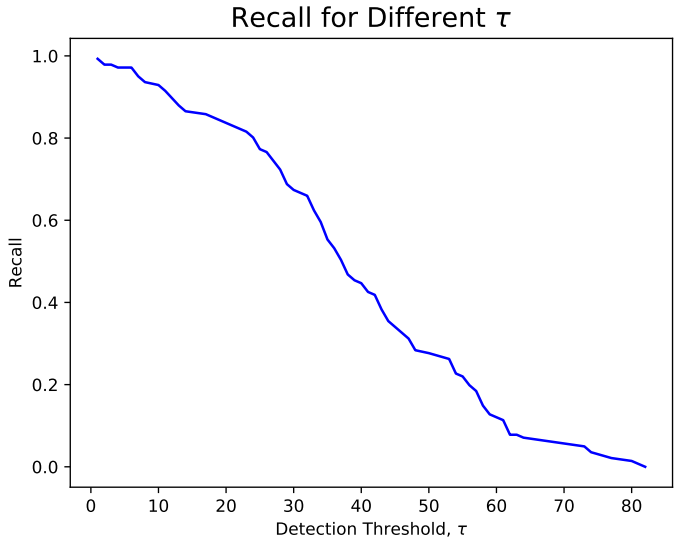

Figure 4.15: Recall obtained for values of τ from 0 to n^2. As τ approaches its maximal value of n^2, Recall drops off to 0.

while larger values favour Pre. This becomes evident when plotting Precision versus Recall for a given threshold τ, as shown in Figure 4.16. In the ideal case, the optimal point on this plot is to get as close as possible to the upper right coordinate of $(1, 1)$.

Due to this tradeoff, it is useful to compute an additional metric, the F_1 score, which is the harmonic mean of Rec and Pre, defined as

$$F_1 = \frac{2}{\text{Rec}^{-1} + \text{Pre}^{-1}} = 2 \times \frac{\text{Pre} \times \text{Rec}}{\text{Pre} + \text{Rec}}. \tag{4.10}$$

Then the choice of τ is the value which maximizes the F_1 score.

In the context of a RAS, anomalies that go undetected (i.e. FN) can ultimately be very costly if it leads to mortalities. These mortalities hurt the

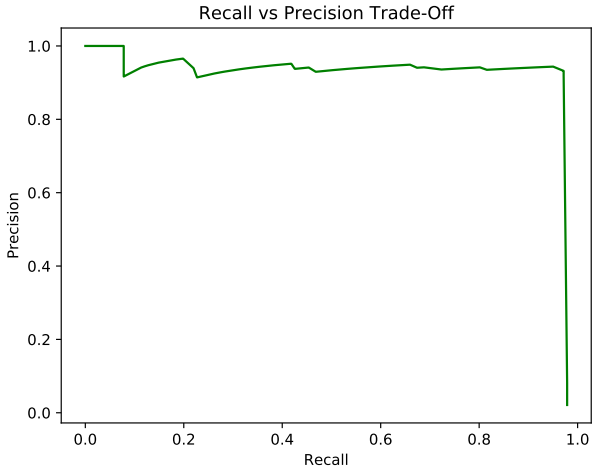

Figure 4.16: Recall vs Precision for some τ.

bottom line of the fish rearing business, both in terms of loss of inventory, as well as increased operator time to clean-up and rectify the damage. On the flip side, false positives while annoying, only trigger an escalation in terms of operator involvement. The only cost this adds to the business is time to investigate the issue until it is determined to be a false positive. Thus an increase in Rec at the cost of a reduction in Pre is a small price to pay to ensure every anomalous event is captured. As such, when comparing models with similar F1 scores, those with better Rec (less false negatives) are preferred over those with better Pre (less false positives).

4.8 Root Cause Identification

The term "root cause" in multivariate timeseries anomaly detection refers to identifying the individual timeseries that contain the anomalous data. That is, the root causes of an anomalous event are the timeseries which are responsible for the anomaly.

4.8.1 Scoring Functions

In order to identify the root causes of an anomaly, the row-wise and column-wise root cause score is computed for each row and column of a given residual matrix. The root cause score in this context is computed using the root cause scoring threshold, γ, which is similar to the threshold θ presented earlier, but is used for root cause identification.

Each timeseries in the residual matrix is ranked by decreasing order of their mean row-wise and column-wise score. For example, when dealing with n timeseries, the j-th timeseries, x_j, is represented by the j-th row and j-th column in the residual matrix. It's score is given by

$$\sum_{i=0}^{n-1} \frac{h_\gamma(r_{ij}) + h_\gamma(r_{ji})}{2}, \tag{4.11}$$

where $r_{ij} \in R^t$ are elements of the residual matrix at timestep t and $h(\bullet)$ is the element-wise anomaly scoring function.

The root cause scores for each timeseries in the residual matrix are ranked

in decreasing order. The top k timeseries are predicted as the root causes of the anomalies. k is technically unknown when used online in a production system, however in this work it is set to $k = 3$ throughout as each anomalous event has at most three root causes per anomaly.

The choices of anomaly scoring functions, $h(\bullet)$, presented in this work are the masked scoring function (which is the scoring function used in the Zhang et al. implementation), with the addition of the linear scoring function and quadratic scoring function, which are newly introduced in this work.

The masked scoring function, $h_{\gamma,\text{mask}}$, is defined as

$$h_{\gamma,\text{mask}}(x) = \begin{cases} 1, & x > \gamma \\ 0, & \text{otherwise} \end{cases} \tag{4.12}$$

where γ is the root cause scoring threshold and x is an element of a residual matrix. In this scoring function, all values greater than γ are flattened to 1, thus it keeps count of scoring elements but loses information relating to how much each element contributes to the overall root cause score.

The linear scoring function, $h_{\gamma,\text{linear}}$, is given by,

$$h_{\gamma,\text{linear}}(x) = \begin{cases} x - \gamma, & \text{if} x > \gamma \\ 0, & \text{otherwise} \end{cases} \tag{4.13}$$

such that all scoring elements (greater than γ) are translated and summed.

Finally, the quadratic scoring function, $h_{\gamma,\text{quad}}$ is given by

$$h_{\text{quad}}(x) = \begin{cases} (x-\gamma)^2, & \text{if } x > \gamma \\ 0, & \text{otherwise} \end{cases} \tag{4.14}$$

such that all scoring elements (greater than γ) are translated, squared and summed.

All three of these scoring functions are used and compared during the analysis of the root cause identification results, to identify which scoring function is most suitable in the context of a RAS and the RAS dataset.

4.8.2 RCI Score

The RCI score is the metric by which root cause identification performance of a trained model is measured.

For each predicted anomaly, the predicted root causes (from the residual matrix) are compared to the ground truth root causes, which is stored in a file during the anomaly injection process (for real anomalies, these are added to the ground truth file manually). For each anomalous residual matrix, every correctly identified root cause adds 1 to the score, while every missed root cause or incorrect guess scores 0. The total is then divided by the total number, k, of root causes for that anomaly. This is called the RCI hitrate of that residual matrix and is expressed in percentage.

Thus, the RCI score for a given model is the mean of the RCI hitrates of

all anomalous residual matrices predicted by that model. It is expressed as a percentage since its range is the closed interval $[0, 1]$.

4.9 Noise Robustness

The property of noise robustness is the level of noise in the dataset (i.e. the SNR) that a model can endure until a sharp decrease in its anomaly detection performance occurs. This property is important to measure, as it indicates whether or not the model can be applied to some dataset ahead of time, simply by analysing the noise floor of that dataset.

This is measured by generating a synthetic dataset without noise, injecting anomalies into this dataset, and subsequently creating multiple copies of this dataset with different levels of noise added to it. In this work, the model's anomaly detection performance is compared for 6 different noise levels, 5%, 10%, 15%, 20%, 25%, and 30% respectively. The model is trained multiple times for each of the datasets, and the resulting average anomaly detection performance of each noise level is compared, in terms of Rec, Pre, and F1.

Chapter 5

Results

This chapter discusses the results obtained from the experiments outlined in Chapter 4. First, the hyperparameter tuning results are examined and demonstrate which hyperparameters had greater effect on the model's ability to learn to recognize normal system states, as well as which combination of hyperparameters yield the model with the best anomaly detection performance, in Section 5.1, followed by an accounting of the best performing models based upon these hyperparameters, in Section 5.2.

Next, in Section 5.3, the anomaly detection performance of the best models are compared in terms of their respective Rec, Pre, and F1 scores.

Section 5.4 compares the ability of the best candidate models to identify root causes of anomalous events.

The results of the robustness to noise experiment are discussed in Section 5.5. Ideas for future work that were brought up during this work are further

elaborated upon in Section 5.6.

5.1 Hyperparameter Tuning Results

An initial exploratory trial tested a broad range of all hyperparameters described in Section 4.4, in order to gather an idea of which hyperparameters had a greater effect on the model's anomaly detection properties. This yielded over 900 possible combinations to test, which was not manageable in the time frame and on the hardware available to conduct this research.

Instead, three separate exploratory trials were conducted to examine the hyperparameters of different components of the model; the first examining hyperparameters related to signature matrix generation, the second recurrent component hyperparameters, and the third convolutional component hyperparameters. During each respective trial, hyperparameters from the other two trials were held constant. Each set of hyperparameters were used to train the model for 5 runs (over 30 epochs each) in order to account for variability in individual runs.

The models in each run were trained on the RAS dataset, containing one real anomaly and four injected anomalies. The anomaly detection performance (in terms of Rec, Pre, and F1) of each trained model was measured and recorded. The average over 5 runs was computed to obtain the mean anomaly detection performance metrics for each combination of hyperparameters.

The results and intuition gained from each trial are detailed in the remainder of this section.

5.1.1 Signature Matrix Hyperparameters

The first hyperparameter trial examined different combinations of window size, window step, and correlation function used in generating signature matrices from the timeseries data. All other hyperparameters were kept constant, using values presented in previous work by Zhang et al.[1]

The best individual trained models obtained for this trial, sorted by highest F1 Score, are presented in Table 5.1. These results represent individually best trained models, however, this is not necessarily representative of which sets of hyperparameters yield better trained models on average. Table 5.2 presents the aggregate performance for each set of hyperparameters, i.e. the mean of the 5 runs. Note, the labelling run-n-m indicates the m-th run of hyperparameter set n, thus the m index is dropped when referring to the mean results in Table 5.2.

Table 5.1: Top 5 models obtained during signature matrix hyperparameter trial, with respect to the anomaly detection F1 score.

Run	**Hyperparameter**			**Scores**		
#	**W Size**	**W Step**	**Corr Fn**	**Rec**	**Pre**	**F1**
run-19-2	$[5, 30, 90]$	1	frac	0.97	0.92	0.94
run-9-2	$[5, 15, 30]$	1	exp	0.96	0.88	0.92
run-10-2	$[5, 15, 30]$	1	frac	0.96	0.83	0.89
run-19-3	$[5, 30, 90]$	1	frac	0.95	0.82	0.88
run-10-3	$[5, 15, 30]$	1	frac	0.82	0.94	0.87

Table 5.2: Best average anomaly detection performance over 5 runs, ordered in terms of F1 score, for the signature matrix hyperparameter trial.

Run Set	Hyperparameter			Mean Scores		
#	W Size	W Step	Corr Fn	Rec	Pre	F1
run-19	$[5, 30, 90]$	1	frac	0.76 ± 0.25	0.73 ± 0.40	0.63 ± 0.36
run-10	$[5, 15, 30]$	1	frac	0.92 ± 0.09	0.54 ± 0.47	0.54 ± 0.46
run-20	$[5, 30, 90]$	1	mscred	0.77 ± 0.11	0.36 ± 0.30	0.40 ± 0.32
run-18	$[5, 30, 90]$	1	exp	0.83 ± 0.13	0.40 ± 0.48	0.38 ± 0.40
run-0	$[10, 30, 60]$	1	exp	0.46 ± 0.30	0.72 ± 0.43	0.38 ± 0.21

Figure 5.1 plots the effect of these hyperparameters values on the resulting anomaly detection performance. In total, there were 27 different sets of hyperparameters, with each point on the scatterplot representing the aggregated results of 5 runs per set.

The top results, with respect to F1 Score, provided in Table 5.1 and Table 5.2, provide some insight into how the signature matrix generation parameters affect the model performance; the F1 Score is the metric of interest; however, Rec and Pre are listed too (Rec being the more important of the two in the RAS context).

The best single model run is **run-19-2** ($F1 = 0.94$), with Rec at 0.97 and Pre at 0.92. The aggregate results of **run-19** provides confirmation on the preference of this hyperparameter set, with a mean F1 score of 0.63. Obtaining such a good result this early on was extremely lucky, as it ends up being the best trained model for the remainder of the hyperparameter trials.

In terms of the window parameters, window sizes $[5, 30, 90]$ performed best both in single model results and aggregated results, with $[5, 15, 30]$ in

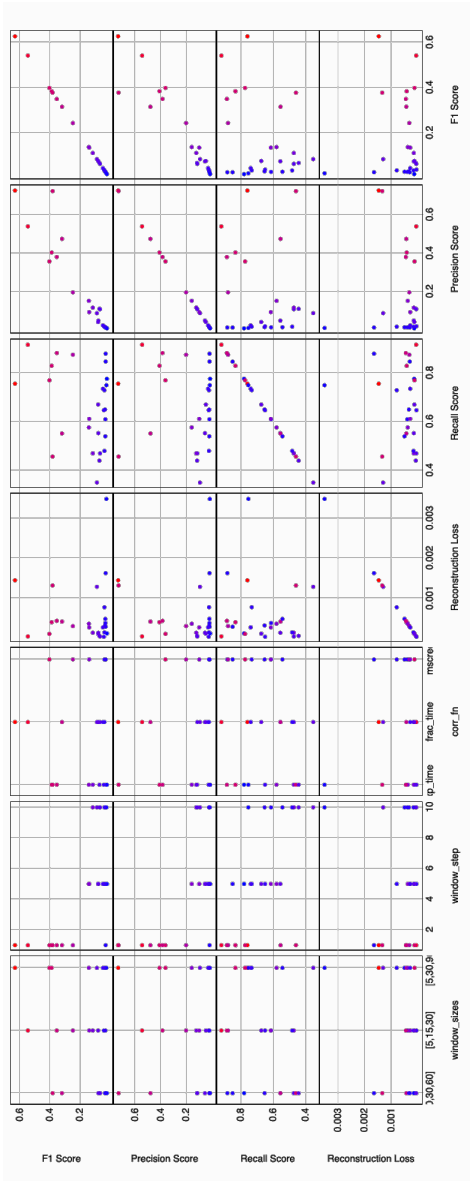

Figure 5.1: Scatter plot of signature matrix hyperparemeters and their respective Rec, Pre, and F1 Scores obtained during testing, as well as Reconstruction Loss during training. Each dot represents one run set, colored on a gradient from red to blue, with red being the best F1 performance run set, run-19, and blue being the worst.

a close second place. The $[10, 30, 60]$ option performed the worst overall in terms of F1 Score (this is seen clearly when referring to Figure 5.1), and is thus dropped from further consideration in later iterations.

The smallest window step value (window step 1) dominates over the larger window steps of 5 and 10, thus the window step is set to 1 going forward.

In terms of correlation function, fractional time (frac) dominates both the single run and aggregate results. However, both exponential and linear time (mscred) still show up in top results, thus they will still be considered in further rounds. Furthermore, the Pearson correlation function showed very poor performance, with its best result yielding $F1 = 0.01$, thus it was dropped from further consideration.

This first hyperparameter trial has already shown that, when compared to the signature matrix hyperparameters used in Zhang et al.[1], the addition of time dependent correlation functions already manifest superior results, at least for the RAS dataset.

5.1.2 RNN Hyperparameters

The second hyperparameter trial was carried out on the recurrent component parameters of the autoencoder: history size (h) and whether or not to bypass the attention layer (on/off). The values used are found in Section 4.4. The signature matrix hyperparameters from the previous sweep were kept constant during this trial, using the values that yielded the best results: window sizes $[5, 30, 90]$, window step of 1, and fractional time correlation

function.

The variability in results was more noticeable in this sweep, thus the number of iterations was increased from 5 to 8 and the training epochs were increased to 35. The best 5 run results of this sweep are presented in Table 5.3 and the aggregated mean results are presented in Table 5.4.

Table 5.3: Top performing models with respect to their anomaly detection performance scores, ordered by F1 score, for recurrent component hyperparameters.

Run	**Hyperparameter**		**Scores**		
#	**h**	**Attn**	**Rec**	**Pre**	**F1**
run-0-3	3	no	0.96	0.81	0.880
run-2-0	5	no	0.74	0.85	0.787
run-2-6	5	no	0.72	0.86	0.785
run-2-2	5	no	0.74	0.83	0.784
run-0-7	3	no	0.67	0.91	0.776

Table 5.4: Mean of aggregrated results of best runs, averaged over 8 runs per hyperparameter set, for recurrent component hyperparameters.

Run Set	**Hyperparameter**		**Mean Scores**		
#	**h**	**Attn**	**Rec**	**Pre**	**F1**
run-5	7	yes	0.62 ± 0.06	0.85 ± 0.04	0.71 ± 0.05
run-4	7	no	0.52 ± 0.16	0.76 ± 0.16	0.60 ± 0.15
run-2	5	no	0.51 ± 0.28	0.72 ± 0.29	0.58 ± 0.29
run-3	5	yes	0.41 ± 0.22	0.77 ± 0.30	0.43 ± 0.26
run-0	3	no	0.79 ± 0.21	0.46 ± 0.48	0.39 ± 0.39

The history size, h, hyperparameter of the ConvLSTM layer was tested for values of $h = 3, 5, 7$. In terms of individual runs, the best performing of these used history size $h = 3$ and $h = 5$. However, the variability of results

in the smaller history sizes led to lower scores among the aggregated results, as seen in Table 5.4.

History size of 7 timesteps led to increased stability in results (i.e. smaller uncertainty), as can be seen by the higher overall score as well as the smaller standard deviation on the mean.

The choice of including or bypassing the attention layer yielded inconclusive results. The anomaly detection performance seems to rely more on history size rather than the use of attention layer on the outputs of the ConvLSTM layers. When looking at results presented in Figure 5.2, using the attention layer improves F1 score by increasing Pre scores, while Rec does not exhibit a clear preference.

In order to achieve optimal hyperparameter selection, history size of 7 and use of the attention layer were selected for the remainder of training, due to its tendency to increase the Pre of the model without noticeably impacting Rec.

5.1.3 CNN Hyperparameters

The third hyperparameter trial looked at the convolutional parameters of the autoencoder; the number of kernels at each layer of the convolutional encoder (e.g. $[16, 32, 64, 128]$ represents 16 kernels at layer l_1, 32 at l_2, etc...), the kernel sizes used at each layer (e.g. $[3, 3, 3, 3]$ represent using a 3×3 kernel at each layer), and kernel stride used at each layer (similarly to kernel size, stride of 2 represents 2×2 stride). Other hyperparameters from previous

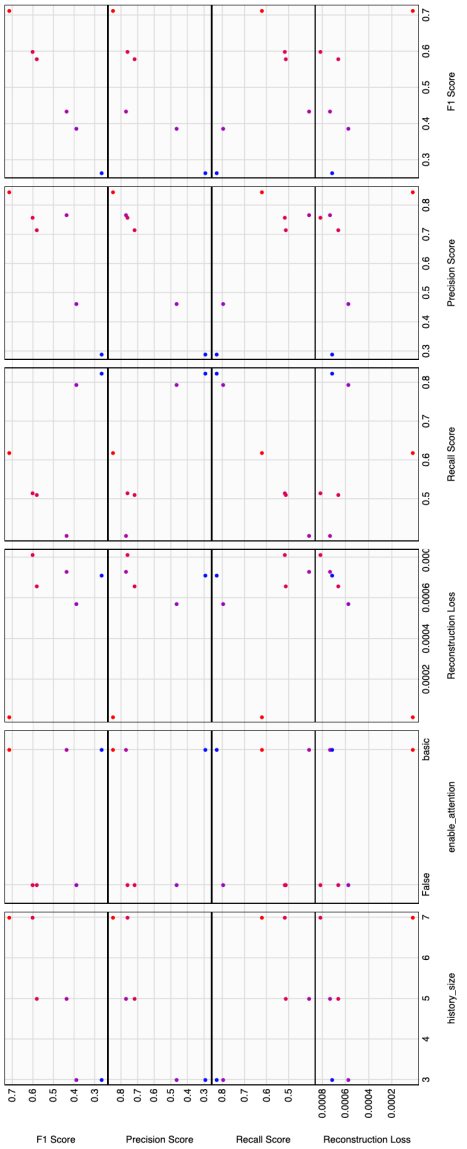

Figure 5.2: Scatter plot of recurrent hyperparemeters and their effect on obtained Rec, Pre, and F1 Scores during testing, as well as reconstruction loss during training. Red dots represent the highest scoring run set in terms of F1 score, while blue is the lowest; every other point is a gradient between these two.

sections were held constant at the same baseline values presented in Zhang et al.[1]

The mean results of the sweep (using values presented in Section 4.4) are shown in Table 5.6 for a mean of 8 runs for each set of hyperparameters, trained for a period of 35 epochs, keeping the best model of each run (tested at every epoch) and averaging them afterwards in terms of anomaly detection performance. The best individual models are documented in Table 5.5.

Table 5.5: Convolutional component hyperparameters for top performing models with respect to their anomaly detection performance scores, ordered by F1 score.

Run	**Hyperparameter**			**Scores**		
#	**# Kernels**	**Kernel Sizes**	**Strides**	**Rec**	**Pre**	**F1**
run-6-3	[32, 64, 128, 256]	[3, 3, 3, 3]	[1, 2, 2, 2]	0.88	0.93	0.91
run-5-6	[32, 64, 128, 256]	[3, 3, 2, 2]	[2, 2, 2, 2]	0.93	0.88	0.9
run-8-5	[64, 128, 256, 512]	[3, 3, 2, 2]	[1, 2, 2, 2]	0.92	0.88	0.90
run-3-2	[16, 32, 64, 128]	[3, 3, 3, 3]	[2, 2, 2, 2]	0.93	0.87	0.90
run-6-1	[32, 64, 128, 256]	[3, 3, 3, 3]	[1, 2, 2, 2]	0.96	0.85	0.90

Table 5.6: Convolutional component hyperparameters averaged over 8 runs for each hyperparameter combination. This represents only the top 5 results out of the total set of 12, ordered by F1 Score.

Run Set	**Hyperparameter**			**Mean Scores**		
#	**# Kernels**	**Kernel Sizes**	**Strides**	**Rec**	**Pre**	**F1**
run-5	[32, 64, 128, 256]	[3, 3, 2, 2]	[2, 2, 2, 2]	0.67 ± 0.26	0.81 ± 0.33	0.63 ± 0.30
run-6	[32, 64, 128, 256]	[3, 3, 3, 3]	[1, 2, 2, 2]	0.83 ± 0.18	0.61 ± 0.42	0.57 ± 0.38
run-11	[64, 128, 256, 512]	[3, 3, 3, 3]	[2, 2, 2, 2]	0.70 ± 0.24	0.72 ± 0.40	0.57 ± 0.32
run-0	[16, 32, 64, 128]	[3, 3, 2, 2]	[1, 2, 2, 2]	0.58 ± 0.21	0.57 ± 0.46	0.48 ± 0.38
run-3	[16, 32, 64, 128]	[3, 3, 3, 3]	[2, 2, 2, 2]	0.77 ± 0.21	0.57 ± 0.46	0.48 ± 0.38

The results in Figure 5.3 indicate a slight preference to using 32 kernels on l_1, 64 kernels on l_2, 128 kernels on l_3, and 256 kernels on l_4, in terms of

F1, Rec, and Pre scores.

The kernel sizes and kernel strides parameters yielded similar results, however slightly better anomaly detection performance is seen with kernel sizes of 3×3, 3×3, 2×2, and 2×2 and strides of 2×2, 2×2, 2×2, and 2×2 in layers l_1 to l_4, respectively.

For this reason, these values are used in the next section, where the model is trained with the best hyperparameters from each trial in order to attempt to get the best performance possible.

5.1.4 Optimized Training

Using the optimal hyperparameters from the previous sections, summarized in Table 5.7, the model was trained 20 times for 50 epochs. The best model in terms of anomaly detection performance, **run-1-9**, scored $Rec = 0.90$, $Pre = 0.87$, and $F1 = 0.88$. The mean performance over the set was given by $Rec = 0.6 \pm 0.1$, $Pre = 0.8 \pm 0.3$, and $F1 = 0.6 \pm 0.2$.

5.2 Best Performing Models

Throughout the hyperparameter trials, the best performing trained models in terms of anomaly detection were saved. The anomaly detection results for these best performing models are summarized in Table 5.8 and visually contrasted in Figure 5.4. These models are used for the analysis in the remainder of this chapter.

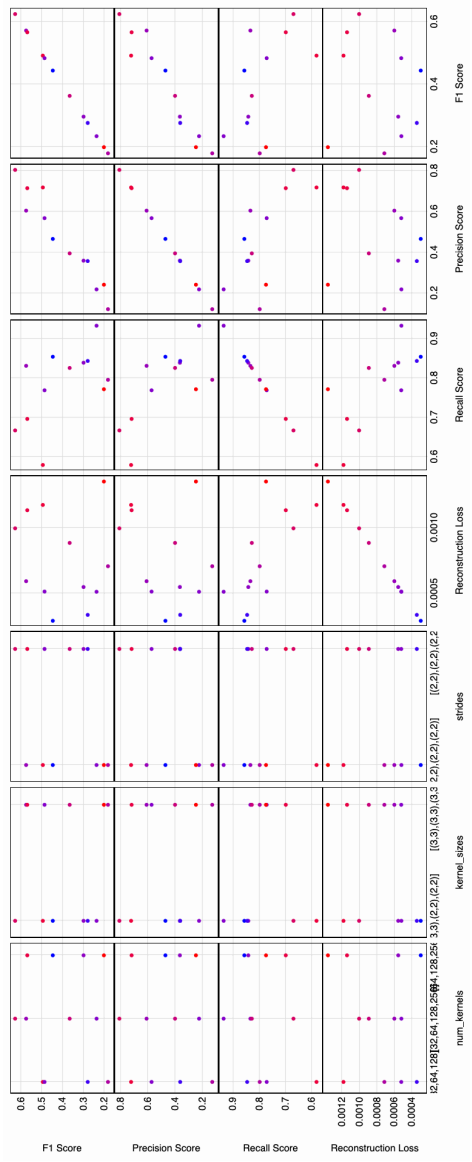

Figure 5.3: Scatter plot of convolutional hyperparemeters and their effect on obtained Rec, Pre, and F1 Scores during testing, as well as Reconstruction Loss during training/validation. Red dots represent the highest scoring run set in terms of F1 score, while blue is the lowest; every other point is a gradient between these two.

Table 5.7: Optimal hyperparameters determined during hyperparameter trials, used for an extensive round of training.

Hyperparameters	
Window Sizes	$[5, 30, 90]$
Window Step	1
Correlation Function	frac
h	7
attn	basic
# Kernels	$[32, 64, 128, 256]$
Kernel Size	$[3, 3, 3, 3]$
Kernel Stride	$[2, 2, 2, 2]$
Best Run	*run-1-9*
Rec	0.90
Pre	0.87
F1	0.88
Mean Performance	*run-1*
Rec	0.6 ± 0.1
Pre	0.8 ± 0.3
F1	0.6 ± 0.2

A compelling question arises. Do these different models, while performing similarly in their anomaly detection scores, detect the same anomalies or different sets of anomalies? In the case of the former, only the best performing model is useful to detecting anomalies in a production system. In the case of the latter, this would imply the use of multiple models in conjunction is beneficial, such that the intersection of sets of detected anomalies across n models would cover the entire set of anomalies present in the dataset.

Table 5.8: Summary of Rec, Pre, and F1 for best performing models. Note that the **Run Set** column represents the name given to each hyperparameter sweep. For example, ras-v3$_{sm}$ refers to the signature matrix hyperparameter sweep on the RAS dataset. The ras-v5 signifies the last run using the optimal hyperparameters discovered during RAS dataset.

Run Set	Run #	Rec	Pre	F1
ras-v3$_{sm}$	run-19-2	0.97	0.92	0.94
ras-v3$_{cnn}$	run-6-3	0.88	0.93	0.91
ras-v3$_{cnn}$	run-5-6	0.93	0.88	0.90
ras-v5	run-1-9	0.90	0.87	0.88

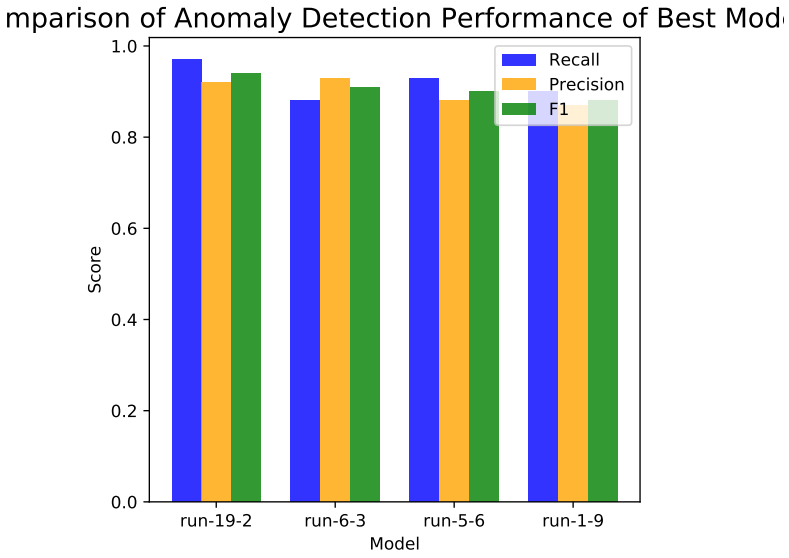

Figure 5.4: Anomaly detection performance scores for the best performing models, where maximum is 1 (100% detection rate) and 0 is minimum.

5.3 Anomaly Detection Results

In this section, anomaly detection results are examined in detail for the **run-19-2** model, as it shows the best overall anomaly detection performance, and is subsequently compared to the results of the other top models listed in

Table 5.8.

A first look at the confusion matrix of **run-19-2**, shown in Figure 5.5, indicates exceptionally low false negative rate (FN), with only 4 instances, as well as only 12 instances of false positives (FP), out of a total of $36,297$ inputs during the testing phase. The true positive (TP) count was 137, with the remainder of inputs being in the true negative's category.

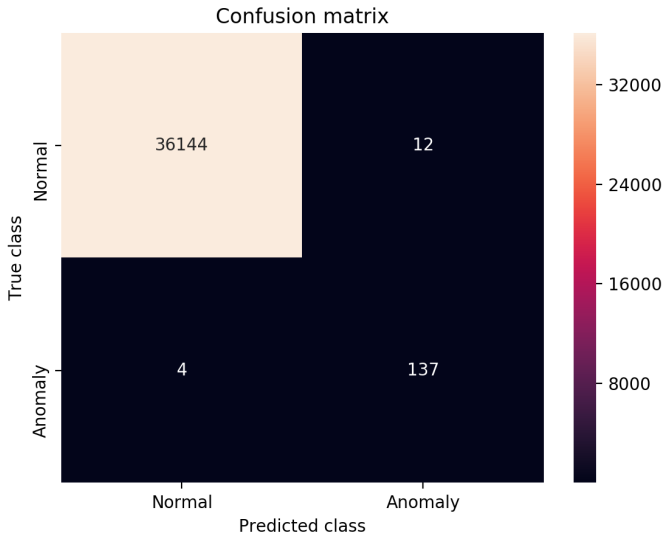

Figure 5.5: Confusion matrix of ras-v3$_{\text{sm}}$ run-19-2.

Figure 5.6 compares the confusion matrix obtained for each of the models listed in Table 5.8, omitting the true negatives count because it is so large relative to the values of interest and overshadow them when plotted. Model **run-19-2** has a significantly smaller FN count compared to the other three models, which is reflected in the Recall score, $Rec = 0.97$.

The greatest concern in the context of a RAS are lowering FN rate. Con-

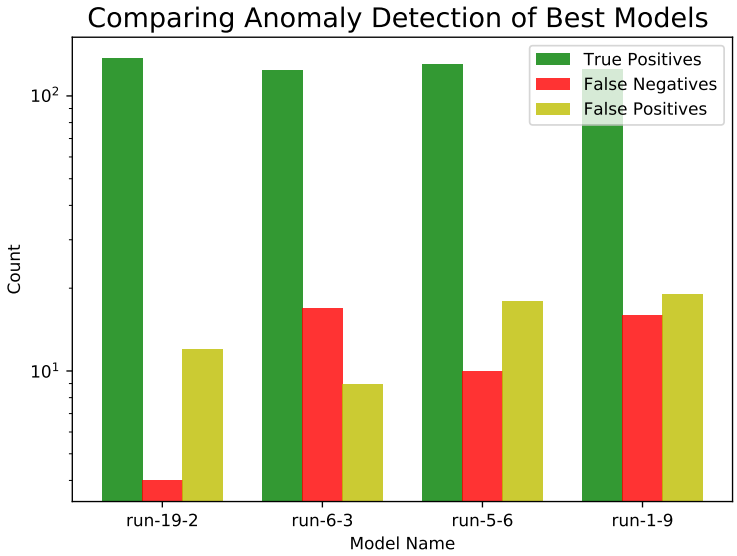

Figure 5.6: Confusion matrix attributes from models listed in Table 5.8 are compared. **run-19-2** has significantly less false negative counts than its counterparts (less than half).

sidering the models presented above, **run-19-2** is the most favorable in this regard.

5.3.1 Identified Anomalies

For model **run-19-2**, all instances of FN occur on the real anomaly, occurring between inputs $9,239$ and $9,301$. The other four anomaly events (injected anomalies) are fully detected, that is, every timestep in the timeframe of the anomaly are accurately predicted as an anomaly. The two first anomalous events for the RAS dataset are depicted in Figure 5.7, showing the success (shown as 1 in the top row and 1 in the bottom row) or failure (shown as 1 in the top row and 0 in the bottom row) at detecting these

anomalous events.

This result is promising, since all FN are confined to one anomalous event, which consists of 7 true positives and 4 false negatives. Therefore, the algorithm would still detect this event as an anomaly in the real world. This is further reinforced by the other 4 anomalous events having a 100% detection rate.

What is not shown in Figure 5.7 are the false positives that were obtained during testing. These were all found to be in close proximity to actual anomalous events. For example, one such true positive occurs right after the first anomaly event, at input $2,800$. This indicates that it is likely that the false positives obtained by the model are due to the model having lag in its recurrent component, which extends into the previous anomaly event.

As such, to answer the previous question of whether or not to use multiple instances of the model to catch all anomalies in a given dataset, it appears that this model could operate on its own given the results obtained from the RAS dataset. This suggests, when used in a production system, the implementation of this model should focus on refining the results of a singular instance of the model, rather than attempts to find n models that, when intersected, cover the entire anomaly set.

To summarize, not every single anomalous timestep is detected as anomalous for a given anomaly, but at least a portion of the timesteps are detected as anomalous. Therefore, each anomaly is successfully detected by the model. The Rec, Pre and F1 scores used to measure anomaly detection performance

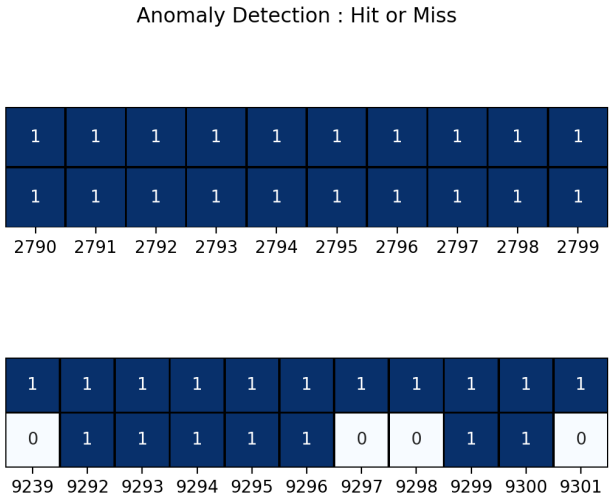

Figure 5.7: Hit or miss diagram of anomaly detection results of **run-19-2**. The top row of each plot signifies the ground truth, y_{true} , while the bottom row signifies what the model predicted, y_{pred}. 0 labels normal data, whereas 1 labels anomalous data. Only the two first anomaly events are depicted, since there are no instances of FN after timestep $t = 9301$.

are defined in terms of *point-based* anomalies and are flawed when it comes to measure *range-based* anomalies, i.e. an anomaly which spans multiple timesteps. Thus, an important improvement to be made to this model in future work would be to implement a range-based Rec, Pre, and F1, as introduced in the previous work of Tatbul et al. [15], which has shown to be an improved method for anomaly detection scoring of timeseries data.

5.3.2 Theta Dependency of Performance

The effect of using arbitrary values of θ on the resulting Rec/Pre/F1 was examined by running the anomaly detection test with values of θ ranging from 10^{-5} to 10^{-1}, as well as including the automatically defined value of $\theta = 0.00223$ and $\tau = 4$ obtained during the test phase (i.e. the value of θ and τ obtained as described in Section 4.6.2).

The results of this scan, shown in Figure 5.8, reveals that the automatically determined value of θ (in the case of **run-19-2**, $\theta = 0.00223$) does in fact yield the optimal $F1$ score. Therefore, using this value in model testing is appropriate. This holds true for each of the other three models tested.

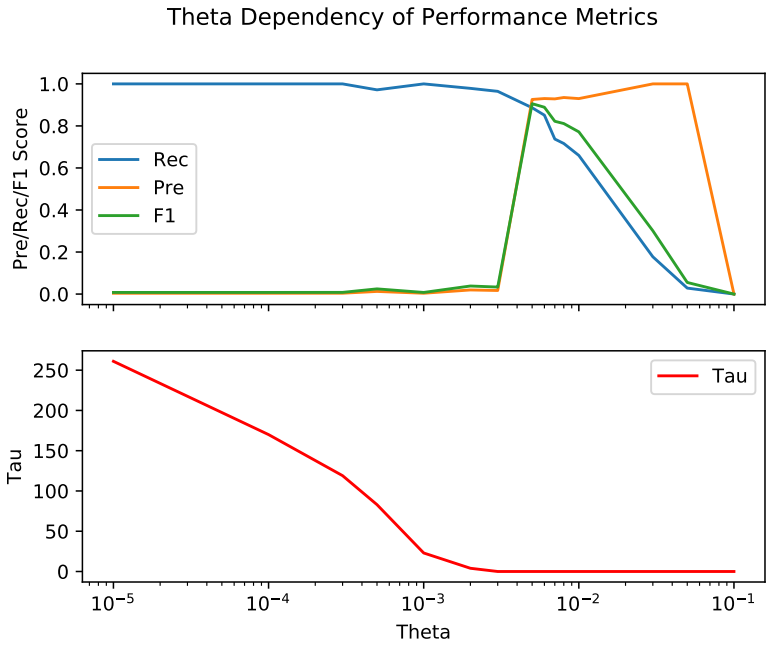

Figure 5.8: θ dependency of Pre/Rec/F1 scores, as well as how this influences the choice of τ. Plot is based on results obtained for **run-19-2**.

Every model performed similarly to the τ plot in Figure 5.8, with τ remaining relatively small (in general, $\tau < 10$, however in the four models presented above, $\tau < 5$).

By establishing these results, the implementations using an automatically calculated θ and τ value are deemed optimal. This removes the necessity of adding these variables among the hyperparameter search-space.

5.4 Root Cause Identification

Root cause identification was carried out on **run-19-2**, adhering to the methodology of Section 4.8. As previously explained, the basis for root cause identification lies in observing differences between the signature matrix which is input into the model and the reconstructed signature matrix it outputs. Figure 5.9 illustrates this difference by comparing an anomalous signature matrix and the corresponding reconstructed matrix obtained from aMSCRED.

The reconstructed matrix has visible reconstruction error relative to the input signature matrix. This reconstruction error is easier to see with the residual matrix, shown in Figure 5.10. Ranking the column-wise and row-wise mean score of the residual matrix, three timeseries (9, 12, 13 respectively) are the predicted root causes for this anomalous timestep, based on their greater reconstruction error.

In this instance, the model accurately predicted the root causes as shown

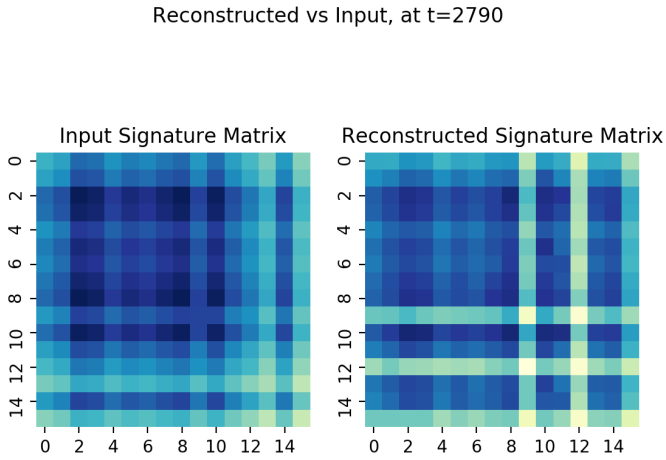

Figure 5.9: The model receives input anomalous signature matrix (left) and outputs the reconstructed matrix (right). Poorly reconstructed rows and columns indicate the root causes of the anomaly. This example is taken from model ras-v3$_{\text{sm}}$ run-19-2, for the 2790^{th} input, which corresponds to the first timestep of the first anomaly.

in Figure 5.11, where the true root causes are highlighted on the input signature matrix, and the predicted root causes are highlighted on the reconstructed matrix from Figure 5.9.

By converting the indices of the anomalous signature matrices back to indices of the corresponding source dataset, the root causes of the anomaly can be plotted on the original timeseries data, as shown in Figure 5.12. The portion of the timeseries that is highlighted in red denotes the portion of the timeseries which is detected as an anomaly. In practice, anomalous events are displayed in this manner on the monitoring system as a visual cue for system operators to observe the nature of the anomaly.

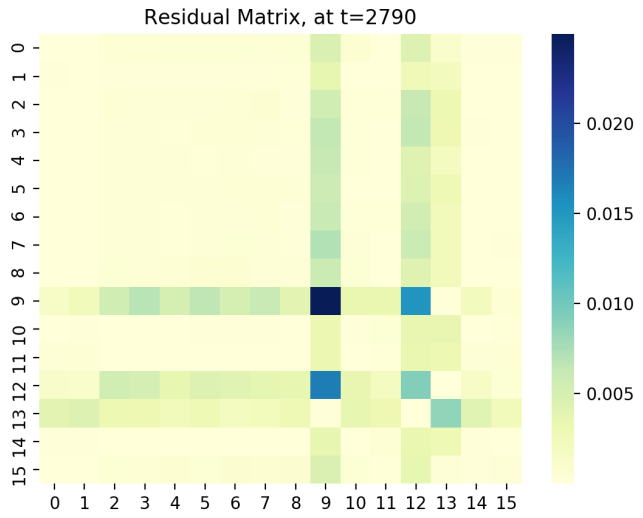

Figure 5.10: The residual matrix obtained for the 2790^{th} input. Pairwise reconstruction error ranges from yellow (minimal error), through green to blue (maximal error). In this example, the greatest source of reconstruction error is timeseries 9, followed by timeseries 12, followed by timeseries 13. Thus, the model predicts them as the root causes of the anomaly.

The RCI hitrate, as defined in Section 4.8.2, for this example is 3 of 3, or 100%. For an example with a lesser RCI hitrate, consider the next signature matrix at $t = 2791$, shown in Figure 5.13.

The RCI hitrate is 1 of 3, or $33.\bar{3}\%$. Examining the residual matrix obtained for this signature matrix, depicted in Figure 5.14, the model only clearly identifies timeseries 9, while the other root causes fade into the background noise. The variability in reconstruction error from one anomalous signature matrix to the next leads to mixed results regarding root cause identification accuracy when looking at signature matrices individually. This demonstrates that even for single anomaly event that is fully detected, the

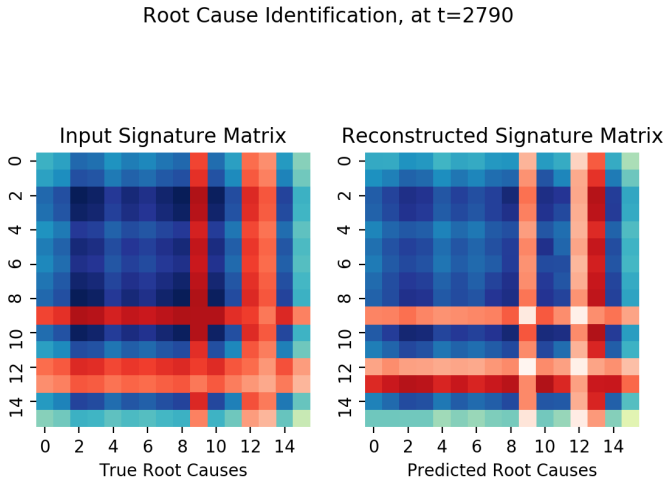

Figure 5.11: Root causes, highlighted in red, predicted by the model (right) match the true root causes of the anomalous signature matrix (left), thus the model has accurately predicted the root causes for the model for this particular signature matrix.

model might not properly identify root causes for each timestep of that anomaly.

For this reason, it is important to compute the RCI score, as defined in 4.8.2, to measure the RCI performance of a model as a whole. The mean RCI score for the model in this example is 72%. The RCI hitrate for each detected anomaly in this example is combined into a histogram in Figure 5.15.

Ideally, the bulk of the histogram should to be mostly in the right-most bin (i.e. 100% detection rate). The best RCI performance is obtained with the linear scoring function. The hitrate of this scoring function is mostly 2 of

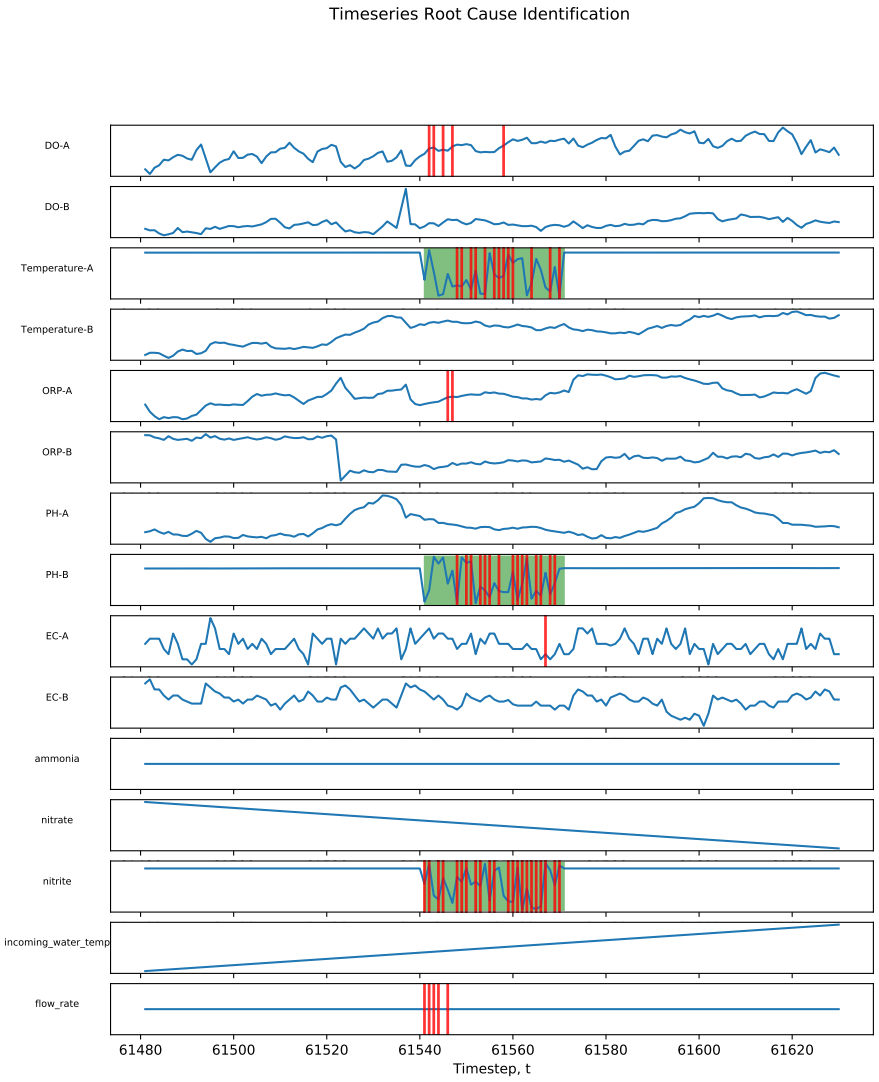

Figure 5.12: Root causes predicted by the model (highlighted in red) are overlaid on the ground truth root causes (highlighted in green) for each timeseries during the duration of the anomaly. Most predictions accurately line up with the ground truth.

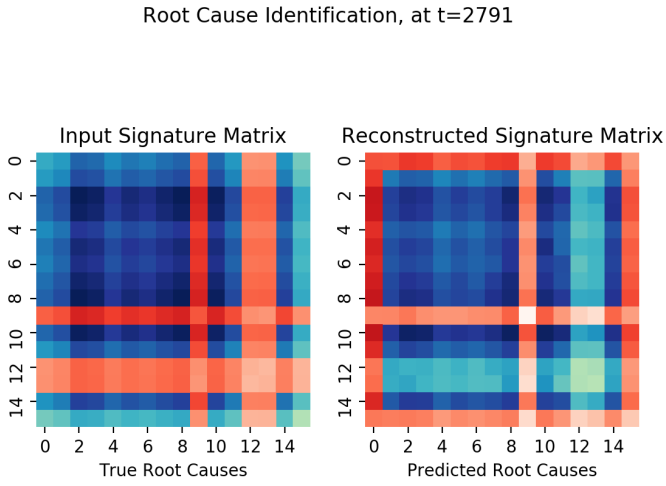

Figure 5.13: An example of where the model incorrectly predicts root causes. The reconstruction error is too low for two out of three of the anomalous timeseries, (below the noise floor) and the model gets confused.

3 (approximately 75% of the time) which is far from perfect, but is reliable enough for technicians to quickly diagnose system issues when an alert is received, knowing that, on average, two thirds of predicted root causes are accurate. The results also show the model does not produce any instances of a hitrate of 0.

5.4.1 γ Dependency of Mean RCI Score

The RCI scores for each of the masked, linear, and quadratic scoring functions were computed and compared for multiple values of γ. The results obtained are plotted in Figure 5.16.

Once γ reaches the same order of magnitude as the noise floor ($\sim 10^{-5}$),

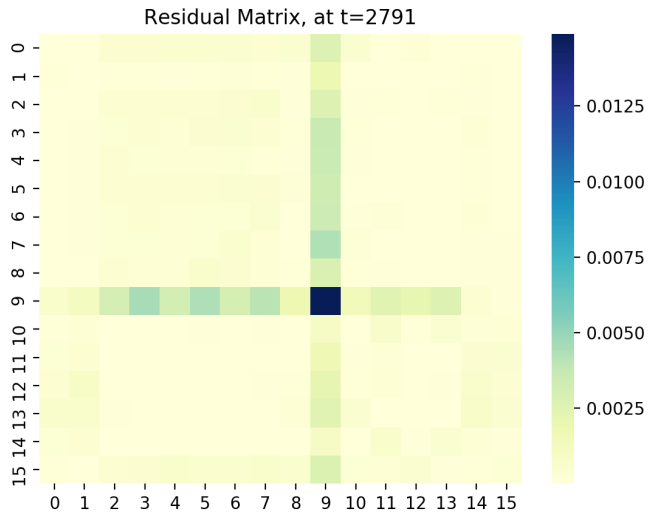

Figure 5.14: The residual matrix for an anomalous signature matrix with a bad root cause identification hitrate. Only timeseries 9 has a detectable reconstruction error. The other root causes disappear into the noise floor of residual matrix (< 0.0025).

the results stay static since it is the same as effectively using $\gamma = 0$; at $\gamma = 0$, the reconstruction loss is identical, regardless of scoring function. In this domain, the linear scoring function delivers the best root cause identification results. On the other end of the spectrum, for values of $\gamma > 10^{-2}$, the scores for each scoring function begin converging and approaching 0.

The best mean RCI score, $h = 72.5$, is obtained using the mask scoring function and $\gamma = 10^{-3}$. However, this value is a local maximum and the mean RCI score drops off rapidly with increasing or decreasing γ.

The linear scoring function achieves nearly identical mean RCI score ($h = 71.8$ at $\gamma = 0$) but decreases as γ is increased from zero. Thus, using the

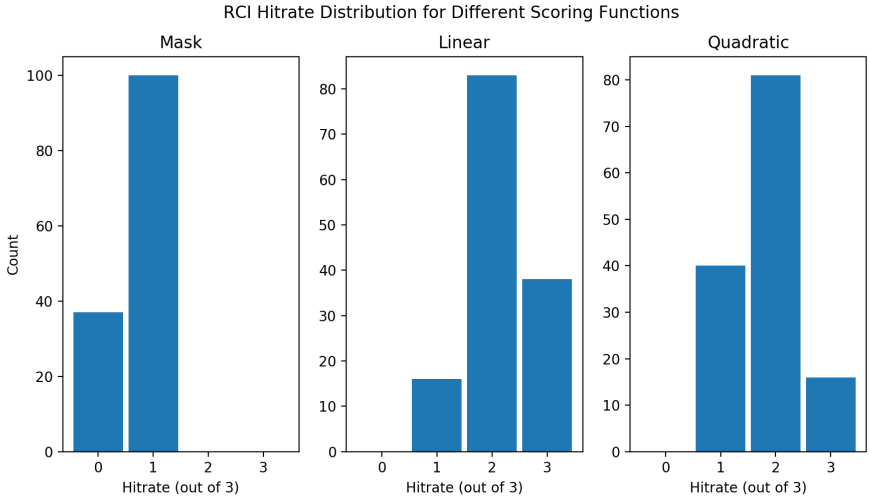

Figure 5.15: RCI hitrate for detected anomalies. Each histogram represents the results for three different RCI scoring functions. They are mask, linear and quadratic scoring functions, respectively.

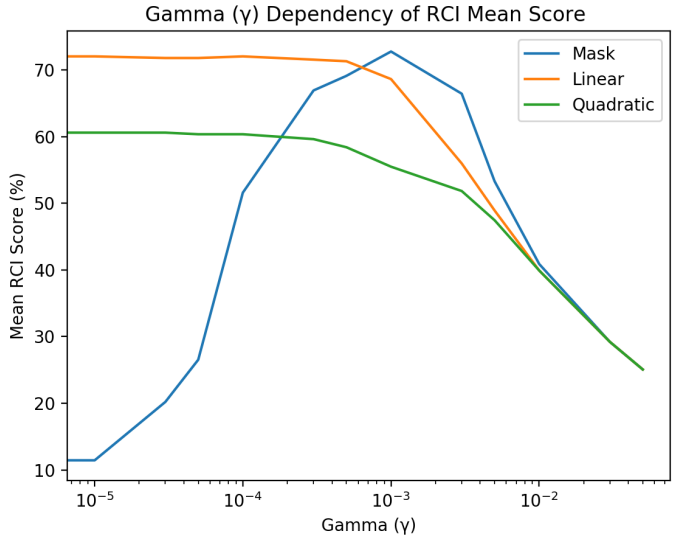

Figure 5.16: Mean RCI scores of the mask, linear, and quadratic scoring functions are computed for multiple values of γ ranging from 10^{-5} to 10^{-1}.

linear scoring function and assuming $\gamma = 0$ eliminates the need to search for the optimal value in γ, reducing overall computation during the RCI process.

The quadratic scoring function follows a similar pattern with respect to γ as its linear counterpart, but consistently worst mean RCI score. This result is expected because any non-zero element in the ranking matrix would only further shrink when squared and disappear quicker below the noise floor, in which case root causes begin to be misinterpreted more frequently than compared to the linear counterpart. This shows that the quadratic scoring function is useless when compared to the linear one and can be discarded.

These results leave an open-ended question of whether or not there could be a set of learnable weights and biases, assigned to each timeseries, such that ranking performance could be optimized by learning the optimal weights and biases during training.

Considering the points laid out above, the linear scoring function is chosen to be used in the production system. It is slightly less accurate than masked function, but removes the requirement of computing the optimal γ and can instead just assume $\gamma = 0$.

5.4.2 RCI Characteristics of Best Models

The mean RCI scores for the best models, from Section 5.2, were computed (using $\gamma = 0$). The results presented in Table 5.9, show similar results across all models.

An interesting observation is that the second run in the table has a higher

Table 5.9: Mean RCI scores of the best performing models. Here the RCI scores are computed using mask, linear, and quadratic scoring functions, respectively.

Run Set	Run #	Mask	Linear	Quadratic
ras-v3$_{\text{sm}}$	run-19-2	24.3%	72.0%	60.8%
ras-v3$_{\text{cnn}}$	run-6-3	23.7%	73.1%	63.7%
ras-v3$_{\text{cnn}}$	run-5-6	23.9%	63.9%	57.3%
ras-v5	run-1-9	24.3%	63.7%	58.1%

mean RCI score than the first, even though the latter had a higher F1 score than the former. However, the Pre score of the second run is higher which unsurprisingly leads to obtaining a more precise RCI score.

5.4.3 Number of Root Causes, k

In this work, the number of root causes, k, is set statically to $k = 3$. This is in part due to the anomaly injection process being hard-coded to only inject anomalies on three timeseries for any given anomalous event, in order to keep the anomaly injection implementation simple and controlled. The shortcoming of this strategy is that it is not practical in the real world, as anomalies could consist of any number of root causes. This shortcoming becomes apparent when looking at the anomaly in the RAS dataset which begins at timestep $t = 45,679$, where two of the three root causes were randomly selected to be timeseries 4. As a result, there are only two true root causes for this anomalous event. However, the RCI process looks for the $k = 3$ most likely root causes and thus wrongly identifies a third non-existing root cause.

In future work, a statistical method could be implemented to try to learn the value of k dynamically on a per-anomaly basis. This could be tested using the same anomaly injection process, however it needs to be slightly modified to choose a random number of root causes for each anomaly, rather than hard-coding it at $k = 3$. This would improve overall RCI performance, by avoiding cases like the two root-cause case mentioned above. With the improved RCI method, it would effectively determine that only two root causes exist and limit the root cause identification to the two respective timeseries.

5.5 Other Results

5.5.1 Noise Robustness

Using seven identical copies of the same synthentic dataset, differing only in noise factor, λ, the model is trained for 15 runs, over 30 epochs, for each dataset. The dataset noise factors were $\lambda = 0\%$, 5%, 10%, 15%, 20%, 25%, and 30%, respectively. The results from 15 runs were averaged to yield the mean and standard deviation of their respective Rec, Pre, and F1 scores, for each dataset, presented in Table 5.10.

The results from Table 5.10 are plotted in terms of Rec, Pre, and F1 in terms of the given noise factor, in Figure 5.17.

The aMSCRED model is fully robust to noise up to $\lambda = 5\%$. The breakdown occurs while approaching 10% noise, where the resulting anomaly de-

Table 5.10: Effect of different noise factor, λ, on the AD performance using the same dataset.

λ	**Rec**	**Pre**	**F1**
%			
0	0.7 ± 0.1	0.76 ± 0.02	0.70 ± 0.09
5	0.6 ± 0.1	0.75 ± 0.03	0.70 ± 0.09
10	0.38 ± 0.04	0.71 ± 0.02	0.49 ± 0.03
15	0.07 ± 0.01	0.5 ± 0.1	0.11 ± 0.02
20	0.03 ± 0.01	0.5 ± 0.3	0.05 ± 0.03
25	0.2 ± 0.4	0.0004 ± 0.001	0.0009 ± 0.002
30	0	0	0

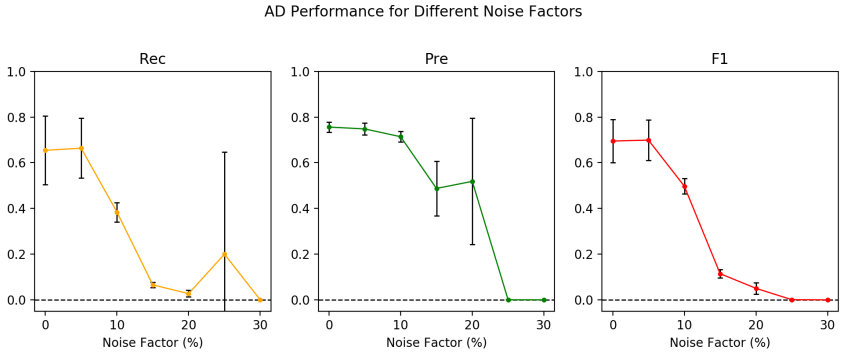

Figure 5.17: Effect of different noise factor, λ, on the anomaly detection performance on identical datasets.

tection scores are already down by approximately a third of their maximum. By the time $\lambda = 15\%$ is reached, Rec and F1 are less than half of the baseline at $\lambda = 0\%$.

This model did not perform as well as expected, considering the implementation by Zhang et al. achieved much higher noise robustness, only experiencing drop-off of performance (i.e. breakpoint) starting around $\lambda = 30\%$.

This discrepancy may be due to the choice of hyperparameters in aM-

SCRED being optimized for the RAS dataset which had a high signal to noise ratio (less than 5%). To improve the noise robustness of the model, a metric for measuring noise robustness could be added to the metrics tracked during the hyperparameter tuning. This is however left as an exercise for future work since it is out of the scope of the objective of this research, since the dataset and system for which this model is implemented is low-noise in nature. An example of how such a metric would work is to arbitrarily choose a desirable noise factor up to which the model should be robust to noise (e.g. $\lambda = 25\%$) and measuring the percentage of drop in Rec/Pre/F1 score at this level from that of the baseline dataset, by creating a duplicate dataset with noise factor λ. Thus, each testing step would occur twice, once on the baseline dataset and once on the noise factor λ dataset.

Another interesting phenomenon is the variation in standard deviation of the Rec, Pre, and F1 scores at different values of λ. It appears that Rec has a large standard deviation at low λ, which decreases up until $\lambda = 20\%$. Pre on the other hand exhibits the opposite behaviour. This implies that as signal to noise ratio decreases in a dataset, the Recall scores across multiple runs of aMSCRED tend to stabilize, which is a peculiar result that should be investigated further in future work.

5.5.2 Selection of Number of Training Epochs

Throughout this research, when deciding how many training epochs are appropriate, the question is actually how many epochs are "good enough".

That is to say, how long does an epoch take in terms of computation time for a given number of training samples (larger datasets take longer)? One could ask, how many epochs are needed to obtain a reasonable reconstruction loss in the shortest computation time?

To answer this question, during implementation of the model and initial hyperparameter sweeps, the number of training epochs was set to 10 as it consistently yielded low reconstruction losses with diminishing returns from adding additional epochs. For the hyperparameter sweeps further on, this number was increased to 35 epochs to yield more precise models. For the RAS dataset in particular, each training epoch takes approximately 90 seconds to complete. This leads to each hyperparameter sweep described in Section 4.4 taking a few days to train.

For finalized model choice however, extra time could be sacrificed to obtain the best possible anomaly detection performance results, since only one set of hyperparameters were tested. Therefore, the model was allowed to train for 100 epochs.

In production deployment of this model, there are long periods of time between new signature matrices being generated (roughly ~ 1 hour) in which the model can be retrained as new data rolls in. Therefore, the default epoch number is set to 25 epochs but this can scale arbitrarily given the availability of GPU compute capacity.

5.6 Future Work

Certain aspects and observations relating to aMSCRED were brought up throughout this work that are of interest to investigate, but lie outside the scope of this research and are left as a starting point for future work and improvement to this model.

The synthetic dataset generator presented in this work is pretty simplistic. The anomaly injection mechanism only has the capacity to inject anomalies as a burst of random noise. Adding multiple anomaly types (e.g. offline sensor, burst error, systematic linear decrease of a signal over time) would add more complexity and the model could potentially learn to recognize and label different anomaly types (e.g. broken sensor, biofilm buildup on probes, etc...)

In terms of signature matrix generation, using learnable weights and biases on the pairwise correlation coefficients obtained from the correlation function instead of using static weights could vastly improve the quality of spatio-temporal information encoded in the signature matrices.

The hyperparameter tuning process started with a small set of possible hyperparameters to search and tested the entire search-space using a grid-search. This method does not scale well when increasing total range of hyperparameters to test. Instead, it is suggested that the hyperparameter tuning process use a random search technique within a much larger hyperparameter search space; larger ranges for signature matrix generation parameters (es-

pecially window size and window step), kernel size and number, as well as history length should be tested. Furthermore, hyperparameters for number of convolutional layers should be made dynamic rather than hardcoded to $l = 4$.

As for measuring anomaly detection performance, the Rec, Pre, and F1 used throughout are optimized for *point-based* anomalies whereas in reality the datasets consist of *range-based* anomalies. Implementing the ranged Pre, Rec, and F1 scores could yield more consistency and less error in their measurement.[15]

Root cause identification has room to be improved, with best results only seeing around 70%. Adding some learnable weights individually to timeseries in the scoring function would possibly increase the capacity of aMSCRED to distinguish between true root causes and false root causes in situations where the reconstruction error of the root causes in the residual matrix is near the noise floor.

Finally, with regard to noise robustness, implementing a metric to measure noise robustness up to some arbitrary (desired) noise factor could be used by the hyperparameter tuning process to optimize model ability to deal with noisy data. The implementation by Zhang et al. had greater noise robustness, therefore it is suspected that aMSCRED's poor performance is due to using hyperparameters that do not favour models that deal with input noise as effectively.

Chapter 6

Conclusions

A novel variant of MSCRED, given the name aMSCRED, was implemented in this work as a means to detect anomalies unsupervised in a RAS. This anomaly detection system is an augmentation of an existing threshold-based monitoring and alerting system that the facility operators interact with daily.

The resulting model was able to outperform the MSCRED implementation of Zhang et al., when comparing Rec, Pre, and F1 scores obtained for the RAS dataset. This dataset contains five anomalies (one real, four synthetically injected). The model was able to detect all five anomaly events, despite producing a non-zero count of false negatives. These false negative counts were part of a larger sequence of inputs that were registered as anomalies. Therefore, in the live RAS system, these anomalies would all be partially detected, which is enough to elicit action from the system operators and

prevent negative consequences to the fish population.

Comparing root cause identification of aMSCRED to the implementation of Zhang et al. showed aMSCRED was able to achieve higher accuracy predicting root causes ($\sim$ 70% compared to $\sim$ 50%) when applied to the RAS dataset. RCI was not tested on the synthetic dataset. This increased RCI performance is in part due to using a new RCI scoring function, i.e. a new way to rank timeseries in terms of their likelihood of being a root cause for any given anomalous signature matrix.

The new signature matrix correlation functions proposed in this work improved overall anomaly detection performance of aMSCRED, as seen during hyperparameter tuning process. When comparing the time dependent correlation functions to the MSCRED correlation function, the time dependent variants yielded greater Rec, Pre, and F1 scores.

The implementation of aMSCRED presented in this work is only a starting point, as many considerations for improvements are presented throughout the work and summarized in Section 5.6. With proper optimization of training and testing times, using this model live in a production RAS would be viable and would be capable of detecting anomalies within a window of 10 timesteps, which in the case of the RAS dataset represents 10 minutes.

Appendix A

RAS Monitoring System

The RAS monitoring system is composed of two main components; the sensor packs (the first component) house the sensors, collect measurements, and relay data to the central monitoring server (the second component), which collects, aggregates, and stores data. A picture of the sensor pack is shown in Figure A.1

The central server is running a custom installation of Zabbix, which is a highly-customizable and feature-rich open-source monitoring platform.[16] In Zabbix, thresholds that define normal operating ranges for each sensor can be defined. When sensor values surpass these thresholds, system operators are alerted via e-mail or push notifications on their mobile devices (depending on the severity of the event, different actions can be taken). Zabbix provides a web-based frontend which can be used from any web browser, on-site or remotely. A custom dashboard on this frontend is configured to show plots of

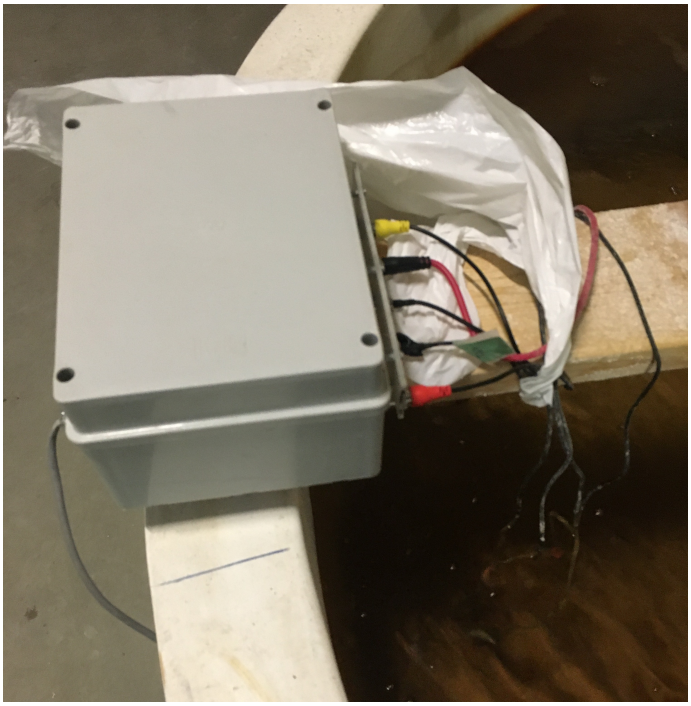

Figure A.1: Sensor pack consisting of 5 submersible probes.

each sensor probe in the system in realtime, and display the latest issues and alerts. A partial screenshot of this custom dashboard is shown in Figure A.2. This dashboard is left open 24/7 on a computer found in the RAS facility, for technicians to inspect daily and get a sense of the state of the system at-a-glance.

The sensor pack uses a Raspberry Pi 3 (Figure A.3), running Raspbian with a Zabbix agent installed, which relays sampled data from the configured sensors every 5 seconds to the central server. A Zabbix Proxy is installed on each sensor pack, allowing local storage of sensor data until relayed to the central server. If communication with the central server is ever lost, sensor

readings queue up locally on the proxy until communication is reestablished. This adds robustness to availability of sensor measurements in the face of communication or partial electrical failures. The unit is powered by PoE, so only a single ethernet cable needs to be run out to each unit. This also improves the safety of the unit, being in a very wet environment, by reducing compromises made to the waterproof enclosure by using a single waterproof ethernet port.

The sensors and respective probes come from the EZO family of sensors from Atlas Scientific. Each probe (Figure A.4) has a corresponding amplification and measurement circuit, to convert the analogue probe signal into a digital value (Figure A.5). Each circuit is connected to the Raspberry Pi USB ports via an FTDI serial to USB adapter (Figure A.6). The advantage of these USB-based boards is that they provide power isolation for electrically sensitive probes (specifically for EC).

The DO, RTD, EC, pH, and ORP probes are submerged around 2 feet in depth and are used to monitor water quality. Additional gas sensors have also been added to one sensor pack which give it the capacity to detect humidity, temperature, CO_2, and H_2S concentration in the air, which is important since the well-water being used releases trace amounts of H_2S.

The full assembly of the inside of the sensor pack is shown in Figure A.7.

Figure A.2: Screenshot of the Zabbix web interface for monitoring sensor readouts realtime. Depicted are the current issues, latest reading for each sensor, followed by a plot of each timeseries.

Figure A.3: The Raspberry Pi 3 is the brain of the sensor pack. It is an ARM-based micro computer which runs a custom distribution of Debian Linux, called Raspbian. It is responsible for gathering readings from the sensors and relaying them to the central server.

Figure A.4: Atlas Scientific EZO probes. From left to right, temperature, ORP, pH, EC, and DO.

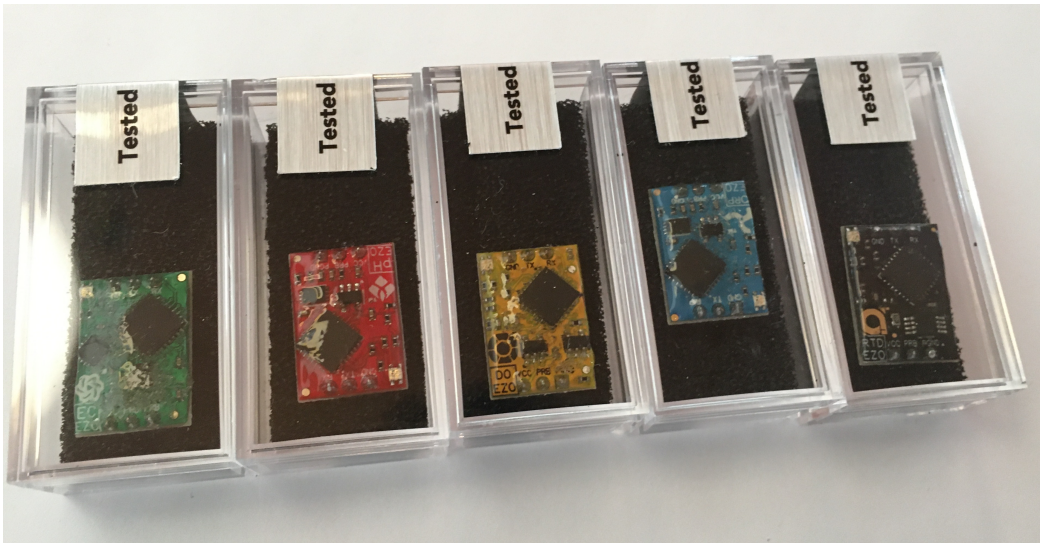

Figure A.5: Atlas Scientific EZO boards. From left to right, EC, pH, DO, ORP, and temperature.

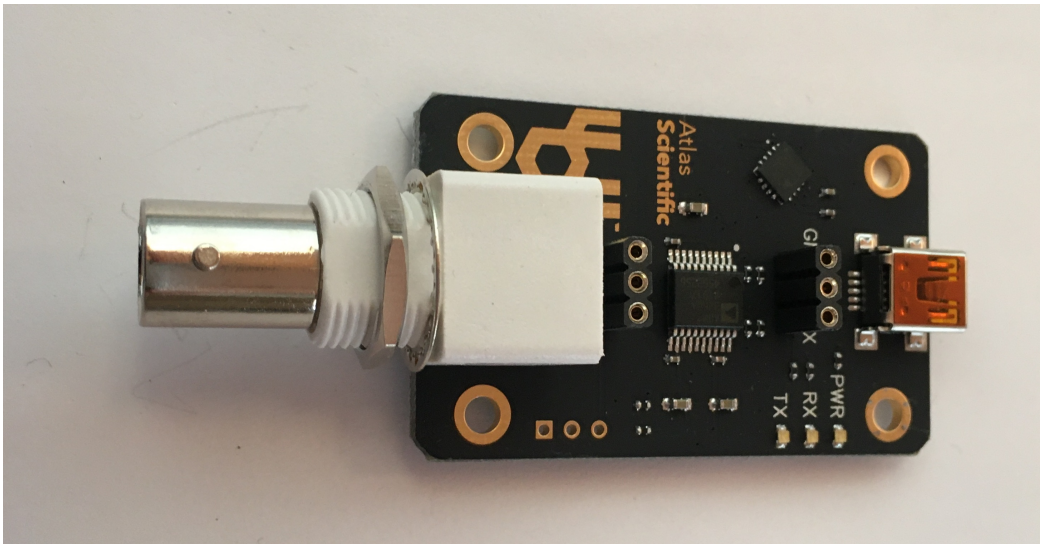

Figure A.6: Atlas Scientific USB carrier board. Interfaces probe (via BNC) to EZO board and digital output is converted via FTDI to the Raspberry Pi, over USB.

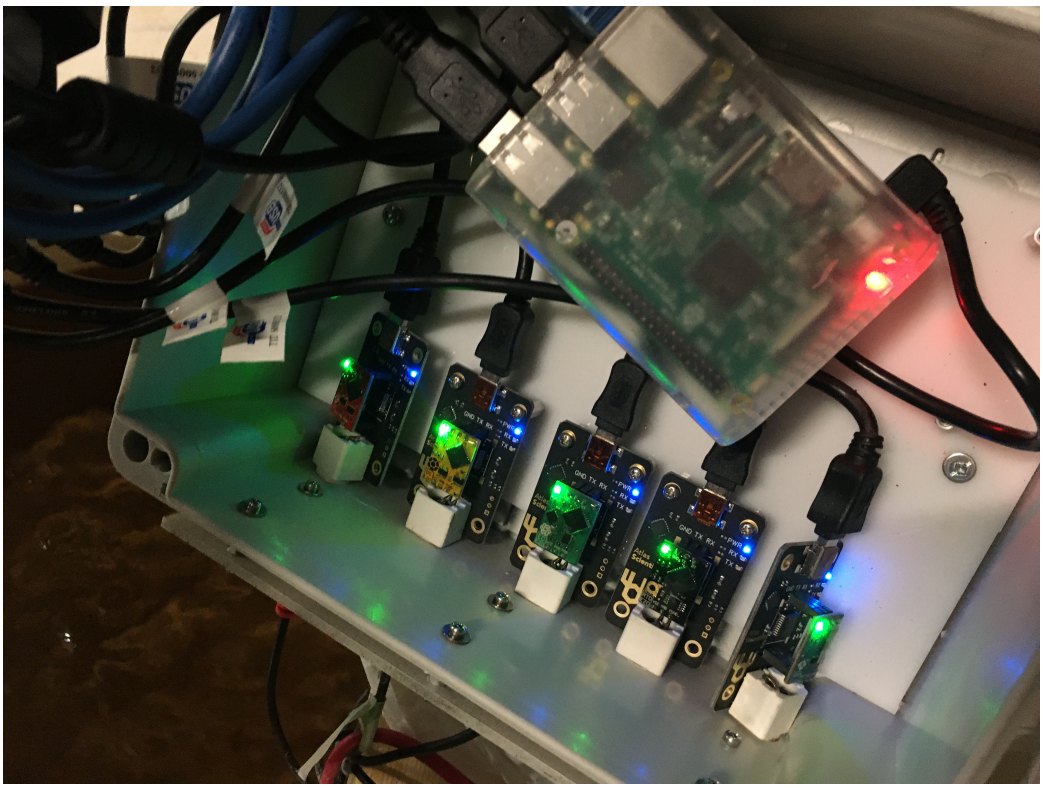

Figure A.7: Full assembly of a sensor pack. The EZO sensor boards interface with the Raspberry Pi via USB carrier boards.

Appendix B

Implementation in Production

To use the aMSCRED model implemented in this work in production, that is to say use it for realtime anomaly detection in the RAS, a few more considerations need to be made. These considerations fall out of scope of the research presented in this work, so are discussed here for reference to those interested.

In this research, the model was trained using dataset of fixed-length, making it only viable to detected anomalies within that period of time. In the production system, the model will need to be retrained every day in order to keep it up to date with changes to the system that occur on larger timescales, on the order of months. An example of such a change are the populations of fish growing larger, leading to increased consumption of oxygen and feed and increased production of waste products. The idea is to constantly train the ML every day so that it catches an evolving picture of the system.

B.1 Data Collection

Every day that goes by without any anomalies occuring is considered "normal" and the data collected from that 24 hour period is appended to the "current" dataset. This means that during this period, no anomalies are detected, no mortalities, no jumpers, and no other significant maintenance made to the system. For example, cleaning out the settling tanks or introducing new cohorts of fish would count as significant maintenance. The size of the "current" dataset can be set arbitrarily long given access to enough compute, but is limited to 60 days in the current implementation. Data collected longer than 60 full days is considered "expired" and is removed from the "current" dataset. Therefore, daily model training sessions only focus on the most recent data.

B.2 Daily Training

After updating the dataset, the model is trained. In order to test the anomaly detection performance of the new model, an arbitrary number of anomalies are injected into the dataset, using the same procedure described in this work. Using the injected anomalies, the newly trained model is put through the same testing routine as described in this work and it's anomaly detection performance is measured. Similarly, the current model is also on this updated dataset. If the new model outperforms the current model, the current model is discarded and the new model is now used to monitor the

system. Otherwise, the previous model remains active.

B.3 Realtime Monitoring

For the remainder of the day, the current active model serves to detect anomalies realtime. Signature matrices are computed every minute using samples obtained during that minute-long period. The signature matrices are queued up in sequential order. Everytime a new signature matrix is computed, it along with the $h-1$ signature matrices preceeding it are fed into the model to obtain the residual matrix.

If the resulting residual matrix is below the anomaly threshold, the system is deemed to be in a normal state and the monitoring system waits to repeat the process all over again.

If the resulting residual matrix is above the anomaly threshold, the monitoring sytem detects this as an anomaly and runs root cause identification routine in order to predict which timeseries contain anomalous data. The monitoring system takes over from there and alerts the system operators of the issue and the predicted causes. Furthermore, data collected from this 24 hour period is not to be added to the "current" dataset and is archived for later analysis.

Appendix C

Tensorflow Model

The model that was researched in this work was implemented in *Tensorflow 2.0* and *Keras*. Shown in Figure C.1 is the structure of the model implemented in TensorFlow. This graph was obtained by opening the compiled keras model into Tensorboard, using the provided graphing tools. The labelling is similar to the labelling used throughout this work, however indexing starts at 0 rather than 1.

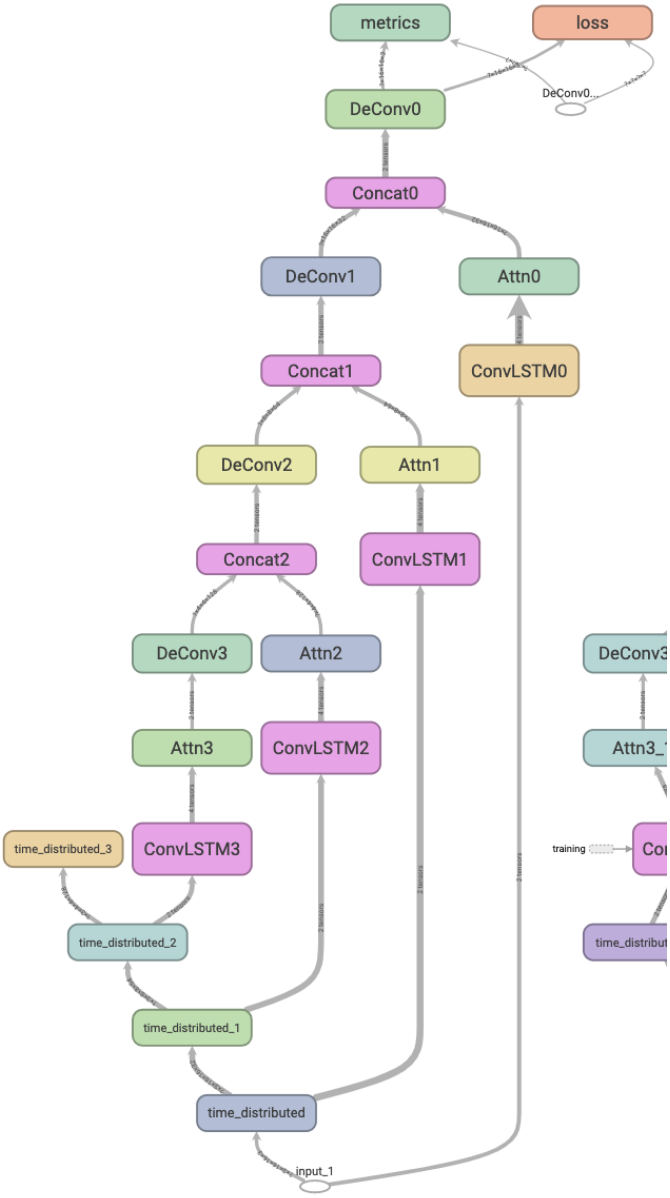

Figure C.1: TensorBoard model representation.

References

[1] Chuxu Zhang et al. "A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data". In: *CoRR* abs/1811.08055 (2018). arXiv: 1811.08055. URL: http://arxiv.org/abs/1811.08055.

[2] Food and Agriculture Organization of the United Nations. "The State of World Fisheries and Aquaculture". In: (2018). URL: http://www.fao.org/3/ca0191en/ca0191en.pdf.

[3] Dongjin Song et al. "Deep r -th Root of Rank Supervised Joint Binary Embedding for Multivariate Time Series Retrieval". In: July 2018, pp. 2229–2238. DOI: 10.1145/3219819.3220108.

[4] Raghavendra Chalapathy and Sanjay Chawla. *Deep Learning for Anomaly Detection: A Survey*. 2019. arXiv: 1901.03407 [cs.LG].

[5] Steven T. Summerfelt et al. "Developments in recirculating systems for Arctic char culture in North America". In: *Aquacultural Engineering* 30.1 (2004), pp. 31–71. ISSN: 0144-8609. DOI: https://doi.org/10.

1016/j.aquaeng.2003.09.001. URL: http://www.sciencedirect.com/science/article/pii/S0144860903000852.

[6] B.A. Barton and B.R. Taylor. *Dissolved Oxygen Requirements for Fish*. Edmonton, Alberta: Northern River Basins Study, Feb. 1994. ISBN: 0662220366. URL: http://www.barbau.ca/sites/www.barbau.ca/files/0-662-22036-6.pdf.

[7] Erik Sandblom et al. "Stress responses in Arctic char (Salvelinus alpinus L.) during hyperoxic carbon dioxide immobilization relevant to aquaculture". In: *Aquaculture* 414-415 (2013), pp. 254–259. ISSN: 0044-8486. DOI: https://doi.org/10.1016/j.aquaculture.2013.07.047. URL: http://www.sciencedirect.com/science/article/pii/S0044848613003852.

[8] Chantelle Penney, Gordon Nash, and Anthony Gamperl. "Cardiorespiratory responses of seawater-acclimated adult Arctic char (Salvelinus alpinus) and Atlantic salmon (Salmo salar) to an acute temperature increase". In: *Canadian Journal of Fisheries and Aquatic Sciences* 71 (July 2014), pp. 1096–1105. DOI: 10.1139/cjfas-2013-0569.

[9] Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *K. Biol. Cybernetics* (1980). DOI: https://doi.org/10.1007/BF00344251. URL: https://link.springer.com/article/10.1007/BF00344251.

[10] Klaus Greff et al. “LSTM: A Search Space Odyssey”. In: *CoRR* abs/1503.04069 (2015). arXiv: 1503.04069. URL: http://arxiv.org/abs/1503.04069.

[11] Xingjian Shi et al. “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting”. In: *CoRR* abs/1506.04214 (2015). arXiv: 1506.04214. URL: http://arxiv.org/abs/1506.04214.

[12] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[13] Günter Klambauer et al. “Self-Normalizing Neural Networks”. In: *CoRR* abs/1706.02515 (2017). arXiv: 1706.02515. URL: http://arxiv.org/abs/1706.02515.

[14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. arXiv: 1409.0473 [cs.CL].

[15] Nesime Tatbul et al. “Precision and Recall for Time Series”. In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 1920–1930. URL: http://papers.nips.cc/paper/7462-precision-and-recall-for-time-series.pdf.

[16] Zabbix LLC. *About Zabbix LLC*. 2019. URL: https://www.zabbix.com/about (visited on 11/01/2019).