

Bachelor Thesis

**Evaluating Sequential Autoencoder Ensemble
for Anomaly Detection**

Phuong Huong Nguyen
10.2023

Lecturers:

Prof. Dr. Emmanuel Müller
Simon Klüttermann, M.Sc.

Technical University of Dortmund
Department of Computer Science
Chair of Data Science and Data Engineering
<https://ls9-www.cs.tu-dortmund.de/>

Contents

1	Introduction	1
2	Related Work	3
2.1	Anomaly Detection	3
2.1.1	Anomalies	3
2.1.2	Anomaly Detection Problem	4
2.2	Autoencoders	7
2.2.1	Overview of the structure	7
2.2.2	Parameters of Autoencoders	8
2.2.3	Anomaly Detection with Autoencoders	9
2.3	Ensemble Methods	9
2.3.1	Overview of Ensemble Methods	9
2.3.2	Anomaly Detection with Independent Autoencoder Ensemble	12
2.4	Evaluation of Anomaly Detection Methods	13
2.4.1	Precision at k	13
2.4.2	ROC curve and ROC AUC Analysis	13
3	Methodology	15
3.1	Sequential Autoencoder Ensemble	15
3.2	Full Sequential Autoencoder Ensemble	17
3.3	Synchronizing Sequential Autoencoder Ensemble	19
4	Experiments and Evaluation	21
4.1	Overview of datasets	21
4.2	The Approach with Autoencoder Model	22
4.2.1	The Architecture of Autoencoder	22
4.2.2	Hyperparameters Analysis	23
4.2.3	Performance of Autoencoder Model	28
4.3	The Approach with Autoencoder Ensemble Models	29
4.3.1	Performance of Independent Autoencoder Ensembles	30
4.3.2	Performance of Sequential Autoencoder Ensembles	31

4.3.3	Performance of Full Sequential Autoencoder Ensembles	34
4.3.4	Performance of Synchronizing Sequential Autoencoder Ensembles . .	36
4.4	Models Evaluation	37
5	Conclusion and future work	43
	List of Figures	46
	List of Tabela	47
	List of Algorithms	49
	Bibliography	54

Chapter 1

Introduction

For a particular dataset, any object that deviates considerably from the remaining data is called an anomaly. Anomaly detection studies the detection of anomalous objects through methods, models, and algorithms based on data [34]. Depending on the availability of labeled data, it could be considered to solve an anomaly detection problem with supervised, semi-supervised, or unsupervised learning. However, having labeled datasets for anomaly detection problems could be very challenging in real-world scenarios, especially when the volume of data becomes larger and larger, which is occurring in this new era of big data. Therefore, unsupervised anomaly detection problems have obtained more attention from studies. One of the most powerful algorithms for unsupervised anomaly detection is autoencoder, which is a neural network with two components - an encoder and a decoder block which are symmetric about a latent layer. While the encoder component compresses an input in the latent space, the decoder tries to reconstruct that compressed data. The difference between the input and its reconstruction defines reconstruction error, which is the key to identifying anomalies in the data. In recent years, the approach to leveraging the strength of various models to generate a more robust model, which is known as the ensemble method, has been explored widely for anomaly detection problems. The most remarkable architectures are independent autoencoder ensembles, which combine multiple autoencoder models in many different ways. However, these architectures tend to produce similar base learners. As a low variance between components, it may not obtain more significant improvements than using an individual autoencoder. To address the problem of variance, we propose various sequential autoencoder ensemble techniques, which are *sequential autoencoder ensemble (SAE)*, *full sequential autoencoder ensemble (FSAE)*, and *synchronizing sequential autoencoder ensemble (SSAE)*. The main point is that prediction errors of previous submodels participate directly or indirectly in correcting errors of the next one. Thus, our techniques can generate autoencoder models with highly different results. This diversity between submodels allows them to learn various aspects of the underlying patterns [14] and support each other to boost the robustness of their ensemble.

The thesis aims to evaluate those proposed techniques by implementing different experiments using nine datasets. The remainder of this work is organized as follows:

- Section 2 explains concepts related to anomaly detection problems. In this section, we also study the autoencoder model in the context of unsupervised outlier detection. The last parts discuss ensemble methods as well as the evaluation of anomaly detection models.
- Section 3 discusses our proposed sequential autoencoder ensemble techniques. In this section, we explain the architecture and algorithm of every variant.
- Section 4 presents experiments conducted in this work and our evaluations of the performance of different models.
- Section 5 briefs the conclusions and suggests some future work.

Chapter 2

Related Work

Anomaly detection (also known as outlier detection) refers to the process of finding objects of a given data that behave very differently from expectation. A common way to decide whether an anomaly detection problem should be solved with supervised, semi-supervised or unsupervised learning is based on the extent of availability of labeled data. In real-world scenarios, having labeled datasets for anomaly detection problems could be very challenging, especially when the data size becomes larger. Hence, most researchers have paid special attention to unsupervised anomaly detection problems. Autoencoder has become one of the most powerful algorithms for unsupervised anomaly detection. Besides, recent works have focused on the approach to leveraging the strength of multiple models to create a more robust model, which is known as an ensemble, for detecting outliers. In this chapter, we study anomaly detection techniques as well as the autoencoder model in the context of unsupervised outlier detection. Moreover, we will also discuss ensemble methods and evaluation of anomaly detection models.

2.1 Anomaly Detection

2.1.1 Anomalies

For a given dataset, any object that deviates considerably from the rest of the objects is called an anomaly. In the context of anomaly detection, anomalies and outliers are the most commonly used terms. Figure 2.1 demonstrates anomalies in a two-dimensional dataset, inspired by [11]. Most data objects concentrate in the areas N_1 and N_2 , and they are normal points that follow a general tendency of the major. However, data points a_1 , a_2 , and some points in the region A_3 are significantly different as they do not share a similar behavior with those in the areas N_1 and N_2 . Therefore, these data objects can be considered anomalies.

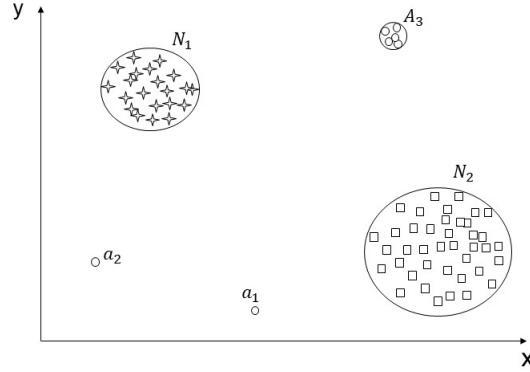


Figure 2.1: An example of anomalies in a two-dimensional dataset

2.1.2 Anomaly Detection Problem

Anomaly detection studies the detection of anomalous objects through methods, models, and algorithms based on data [34]. Most of the anomaly detection techniques deal with a particular formulation of the problem [11], which is defined by various features such as the nature of the input data, the extent of availability of labeled data, the output of anomaly detection and so on. We will discuss some aspects of an anomaly detection problem.

a. Nature of Input Data

Input data is a collection of observations (also referred to as object, record, point, vector, sample, instance, or entity [11]) regarding a specific domain. The nature of input data plays a key role in determining which anomaly detection technique should be used to solve the problem. For instance, for statistical methods, continuous and categorical data require normally different statistical models. For nearest-neighbor-based methods, which measure should be used to identify the distance between clusters depends on the type of input data. With complex data types such as images, video, audio, text, graphs, multivariate time series, and biological sequences [34], classic approaches for anomaly detection problems like Principal Component Analysis, Support Vector Machine, Nearest Neighbor Algorithms, and so on are rarely taken into account. Since a meaningful representation of the data is important for the success of detecting outliers in complex and high-dimensional data [34], [7], deep learning has proved its ability to handle such data types.

b. Data Labels

In terms of anomaly detection, labels of data objects indicate whether an observation is normal or abnormal. The availability of labeled datasets used for anomaly detection problems is usually a major challenge in reality. For instance, in areas where catastrophic events would be anomalous observations that should be detected, it is mainly not possible to get labeled datasets for training models. In many other cases, having accurately labeled datasets is really expensive and time-consuming, since labeling data objects in a specific field is normally done by human experts in that domain [11]. Therefore, besides various approaches to categorizing anomaly detection problems, the available extent of data labels is a standard and most commonly used way to divide the former into supervised, semi-supervised, and unsupervised learning.

- Supervised Anomaly Detection:** An outlier detection problem can be accessed with supervised learning when a dataset used for training a model is completely labeled [34]. In this case, the entire observations in a given dataset have been denoted as normal or anomalous, and thus this learning problem is typically solved with supervised binary classification. The model will learn to determine whether a data point belongs to normal or abnormal classes. One significant aspect to makes the supervised anomaly detection vary from normal classification tasks is the imbalance in classes. As outliers are often very rare in a dataset, the distribution between the normal and abnormal classes will be very skewed, and hence the optimization of classification accuracy may not be meaningful [3]. Another considerable issue is that a given labeled dataset may not capture fully all types of anomalies during annotation [22]. It leads to the limitation in detecting unknown anomaly types of the supervised model. Because of some related issues and possible high costs involving labeling data, the supervised anomaly detection is the most uncommon approach in practice.
- Semi-supervised Anomaly Detection:** Semi-supervised methods for anomaly detection tackle such scenarios that only a few objects in a dataset are labeled, whereas the remaining is unlabeled [21]. Since labeling might require domain experts, obtaining a completely labeled dataset may take a high cost and time, especially when the data size increases. Also, abnormal observations are infrequent in many fields and may be difficult to be labeled. Therefore, it is more common to solve anomaly detection problems with semi-supervised methods than supervised ones. For instance, a typical approach in semi-supervised techniques is to train a model for normal data objects, when some labeled ones are available. The model trained with normal data points is used to detect anomalies in the test dataset by classifying the objects not fitting the former as outliers [21]. However, a model for detecting anomaly trained on a dataset with only few labeled abnormal objects performs mainly just a

little accuracy as these some outliers may not represent all the possible anomalous types. Thus, the most priority solution is to solve anomaly detection problems with unsupervised methods.

- **Unsupervised Anomaly Detection:** An anomaly detection problem should be approached with unsupervised methods in case only unlabeled data is available for training a model [34]. Unsupervised anomaly detection techniques have been the most widely used approach since they do not require knowledge about data labels, which is really difficult to fulfill in real-world scenarios. Instead of learning from a given labeled dataset to classify new data points, an unsupervised model expects that most objects will share some similar patterns while anomalies are far away in feature space from the majority, and thus they become detectable through deviations from the model [21], [34]. Nevertheless, this assumption is not always correct. For instance, normal observations in a dataset may be distributed diversely and hence do not follow strong patterns. Generally, unsupervised methods are dynamic as they depend mostly on the nature of training datasets. In recent years, many unsupervised techniques have been proposed to deal with the fact that data are becoming larger, more complex, and high dimensional. The recently explored methods can be categorized into shallow and deep neural network methods, in which the former tends to carry better interpretability, whereas the latter can be good at tackling large, high-dimensional data [22].

c. Output of Anomaly Detection Methods

Another property regarding anomaly detection problems that should be considered is how a trained model reports possible outliers. Generally, the outputs of anomaly detection methods can be labels or score types [11].

- **Labels:** The outputs generated by supervised will be classified as normal or abnormal objects. Although this type makes a clear conclusion about the anomaly of each data point, it limits experts' abilities to re-evaluate and give a suitable choice regarding outliers in a specific dataset. This point could be crucial in many domains where the boundary between normal and abnormal data is unclear or the tolerance to accept a possible outlier as a normal object can be adjusted flexibly.
- **Scores:** Usually, semi-supervised and unsupervised anomaly detection methods return the outputs as a list of anomaly scores. In particular, the former assesses an anomaly score for each data object based on the extent it is considered an outlier [11]. Having a ranked list of anomalies gives domain experts more flexibility to finalize the outliers of the dataset. They can decide to analyze the top few anomalies or apply a specific threshold for selecting the most relevant anomalies. Scoring-based anomaly

detection methods play a role as supporters for domain experts to make their final decisions about anomalies in a dataset. Evaluations of domain experts may be very important in anomaly detection problems because of their sensibility.

2.2 Autoencoders

The study [35] has first introduced Autoencoders as a neural network for reconstructing an input by learning in an unsupervised manner a meaningful representation of the data [6]. Originally, autoencoders were used for dimensionality reduction or feature extraction [26]. Recent researches have shown that autoencoders have significant capability for unsupervised anomaly detection.

2.2.1 Overview of the structure

An Autoencoder is a neural network trained to reconstruct its input to its output. Instead of attempting to copy its input perfectly, the autoencoder learns valuable properties of the training data because of being forced to prioritize which aspects of the former should be copied [20]. The general structure of an autoencoder comprises two components - an encoder and a decoder network which are symmetric about a latent layer. A latent layer or latent space is a single layer of neurons that represent the reduced representation [14]. The encoder and decoder blocks of a simple autoencoder can be defined in equations below [4], [13]:

$$h = \sigma(W_{xh}x + b_{xh}) \quad (2.1)$$

$$z = \sigma(W_{hx}h + b_{hx}) \quad (2.2)$$

$$L(x, z) \quad (2.3)$$

Where sigma is the non-linear activation function, and W and b are the weight and bias parameters of the neural network.

To obtain useful features, the encoder component in equation (2.1) constrains the internal representation h to have a smaller dimension than input vector x through non-linear transformation via an activation function [13]. On the opposite, the decoder in equation (2.2) tries to revert the original input with the same transformations in the encoder. The learning process attempts to minimize the loss function, as in the expression (2.3), which penalizing the reconstructed output z for being dissimilar from x [20]. This loss demonstrates the reconstruction error whose value is the base for detecting anomaly of data. Data points with a substantial difference from the majority will have higher reconstruction error.

2.2.2 Parameters of Autoencoders

Obtaining an outperformed Autoencoder requires the fine-tuning of various architectural features and hyper-parameters. Below are some decision criteria and parameter selection decisions that should be taken into account.

- **Layers:** The layers of an autoencoder are divided into the encoder and decoder blocks and regulate the depth of the network. The more the non-linearity and complexity in the data, the higher should be the number of layers to extract the important representations from such data [36]. However, deeper networks are more difficult to train. Hence, a good trade-off between modelling capability and training challenges should be considered.
- **Latent Space:** A latent space contains the compressed representation and is the most crucial hyperparameter for tuning autoencoders. The latent layer limits the information passed through from the encoder and prevents therefore the network from memorizing the input and overfitting on the data [5]. Hence, the smaller the latent size, the lower the probability of overfitting issues. But it would lead to a slipping out of important information in the data.
- **Activation Function:** In artificial neural networks activation function is used to transfer an input signal of a node to an output signal which in turn is fed as input to the next layer in the stack [38]. Activation functions are very important as they define how successfully a model learns from the data. In most cases, the activation functions for hidden and input layers are various from that of the output layer.
- **Loss Function:** The loss function computes the difference between the expected and the predicted output of a neural network. A neural network's prediction accuracy can improve significantly with better loss functions, even when the network architecture does not change [44]. The mean square error (MSE) is the most common loss function in deep learning and has shown its' outweighed efficiency in our experiments. For a dataset D with n data points, the mean square error is defined as:

$$MSE = \frac{1}{n} \cdot \sum_{j=1}^n (y_j - \hat{y}_j)^2 \quad (2.4)$$

where y_j is a data point in D and \hat{y}_j is its' predicted value.

- **Batch-size:** The batch size, which is the number of samples used in every epoch to train a network [29], is one of the main hyperparameters that need to be tuned. While using a large batch may result in a degradation in the quality of the model [30], too small batches may take a long time to converge. It is common to use a batch size of 32-512 samples in general practice [30].

- **Epochs:** The quantity of epochs defines the number of times that a model works through the entire training dataset [10]. There is no limitation about the number of epochs, but an epoch of hundreds or thousands is commonly used in practice. A large epoch allows a model to learn till the error from the model has been sufficiently decreased, although it also requires more computational cost and is not always in direct ratio to the quality of the model.
- **Dropout Regularization:** Dropout, which is a technique for addressing overfitting and combining exponentially many different neural network architectures efficiently [39], is only applied during training time. By dropping randomly some nodes out from the given layer, the neural network is sampled to a thinned one. The probability of eliminating a node in a particular layer is a hyperparameter need to optimize.

2.2.3 Anomaly Detection with Autoencoders

In order to reconstruct an input, an autoencoder compresses the former at first in the encoder layers before reproducing it in the decoder layers. The unperfect mapping input to output of the autoencoder leads to the difference between the input and its reconstruction, which defines reconstruction error, the key to identifying anomalies in the data. Since autoencoders learn to reconstruct data based on the distribution of the majority of data [13], data points that differ significantly from the bulk have higher reconstruction errors. Any data point with the reconstruction error exceeding a certain threshold is labeled as an outlier.

2.3 Ensemble Methods

The idea of training a set of different models separately and then combining them in some way to vote on the predictions of new data points in supervised and unsupervised learning problems is known as ensemble methods. These methods leverage the strength of a committee of learning models to obtain more accurate predictions. In the literature, ensemble learning has been explored widely and attained more and more attention from researchers in recent years.

2.3.1 Overview of Ensemble Methods

Different models trained on the same dataset or fitting various partitions of a dataset on a model architecture may lead to varying performance since each one might learn a certain aspect of the data. Therefore, an ensemble combining the strength of various base learners is often much more robust than the individual models included. To construct a good ensemble, each base learner should be accurate and diverse [24]. The accuracy means each base learner has an error rate better than random guessing on new data, while the diversity

ensures all models will not make the same errors on the test examples [17]. Training an ensemble contains two steps: creating a set of base models from the training data, and then combining them to get the final composite predictor [25]. In the literature, many different variants of the ensemble have been proposed. There are some popular methods in supervised and unsupervised learning as follows:

- **In Supervised Learning:** The most well-known ensemble methods are bagging, boosting, and stacking.
 - **Bagging:** Bagging (short for **Bootstrap aggregating**) is a method that produces several versions of a model by training them with different datasets constructed from an original dataset and then uses these variants to form an aggregated predictor [9]. For instance, to train an ensemble of K base models, bagging constructs at first K different training sets by sampling randomly with replacement from the original training set. That means each assembled set has some missing examples replaced by duplicating the others in the original dataset. Then, by fitting diverse constructed training sets on a particular model, bagging generates multiple learning versions. The aggregated result is obtained by averaging over the base models when the outcome is numerical or voting for a plurality in case the outcome is a class
 - **Boosting:** Boosting [19] is one of the most powerful ensemble methods which combines many weak learners, whose error rate is only slightly better than random guessing, to generate a robust predictor. Like bagging, boosting manipulates an original dataset to generate multiple base learners, but in a different way. Instead of sampling independently training sets from the original dataset as bagging, boosting modifies the training set by applying weights to each of the training examples. For instance, to train an ensemble of K base learners, the AdaBoost algorithm, developed by Freund and Schapire [19], sets weights $w_i = 1/n$ to all training observations at initialization, where n is the number of observations. For each iteration $k = 2, 3, \dots, K$, the previous base learner contributes to the modification of observation weights in the next iteration. Generally, boosting algorithms work by training sequentially a weak learner with modified versions of the training data, therefore generating a sequence of weak models. The final prediction is a combination of the entire weak learners' predictions in some way. There are some popular algorithms using boosting strategy like AdaBoost (Adaptive Boosting), XGBoost (Extreme Gradient Boosting), and Gradient Boosting.
 - **Stacking:** Stacking can be used as a technique for not only combining learning models but also improving a single algorithm [43]. In the context of ensemble methods, stacking attempts to find the best combination of predictions made by

base models concerning a training dataset. Unlike bagging, in stacking, the base learners are typically various but trained on the same dataset. The predictions of the whole base learners can be cast as inputs in the next step, training a meta-model that learns how to combine these inputs to make a better prediction. Although a meta-model can use any learning method, it is common to use linear regression for regression problems and logistic regression for classification tasks.

- **In Unsupervised Learning:** Ensemble methods have become increasingly popular in unsupervised learning in recent years. Unlike in supervised learning, an unsupervised ensemble obtains the predictions from base models with unknown reliability, because they are trained on a dataset of unlabeled examples. The possible conflict of base learners' predictions has made the step of combining their results more difficult and flexible according to each specific problem. One popular unsupervised ensemble method has been proposed by Dawid and Skene [16], which assumes that all classifiers are conditionally independent. They proposed to estimate the accuracy of classifiers by the Expectation-Maximization algorithm, which only ensures the convergence to a local optimum [37]. There are still some limitations in the model of Dawid and Skene. Firstly, the assumption of perfect diversity between classifiers is commonly unrealistic in real-world scenarios. Secondly, the assumption of all instances' equality leads to each classifier has the same probability of error [28]. Despite its limitations, the model of Dawid and Skene is still considered a pioneer in unsupervised ensemble learning and motivated the further research such as [33], [40], [18], [32].

Overall, the term ensemble method can be generalized as any approach which combines the outcomes of either independent or dependent executions of machine learning algorithms [2] and hence can be categorized by component independence as below:

- **Independent Ensemble Learning:** An ensemble is considered independent when its base learners are trained independently. That means the execution of a particular base model does not depend on the result of the other in the ensemble. No matter how the components of an ensemble are built, they can be trained by using different algorithms or fitting a model with various variants of a dataset, the prediction of one base learner does not affect that of the other. There is no regulation in determining the final prediction of an independent ensemble. It could simply choose one of the base models' predictions or be a combination of the whole predictions in some way. Bagging and stacking algorithms are a few of many classification ensembles whose components are independent of each other.
- **Sequential Ensemble Learning:** A sequential ensemble contains multiple base learners that depend on one another. The idea is that the outcome of one or many previous base models has an effect on the next one to some extent. With this ap-

proach, one base model is expected to improve accuracy by learning the experiences of its predecessors. Similar to the independent ensemble method, the final prediction of a sequential ensemble can be inferred in different ways such as using the outcome of the last base model or combining the results of all base learners. Boosting can be categorized as sequential ensemble learning because the execution of a particular base model depends on the outcome of previous ones [2].

2.3.2 Anomaly Detection with Independent Autoencoder Ensemble

Whether anomaly detection should be approached by supervised, semi-supervised or unsupervised learning depends on the availability of labels about the anomalousness of data points in a training set. In practice, it becomes more difficult to get labeled datasets as data size continues to increase. Therefore, unsupervised outlier detection has received more attention from studies. Autoencoder is considered an outstanding approach for anomaly detection because of the ability to generate a reduced representation of the data [14]. Specifically, outliers are not normally represented as accurately as normal points in a reduced representation. As a result, the reconstruction error of anomalous points will be larger than that of the other. Besides tremendous performance, there are some limitations to using an autoencoder model for outlier detection. Since a given autoencoder model may learn to perform well on a particular dataset, but may not behave well on others. It may also happen that just by a slight adjustment in the size of the latent layer, a given autoencoder will learn other aspects of a given dataset and thus make different reconstruction errors. Hence, it could be optimistic to assess a point as an outlier base on the outcome of an individual model only. Recent works have proposed autoencoder ensemble methods to increase the robustness of learning models while decreasing the bias of the underlying models as well as the variance of the resulting model [12], [8]. An autoencoder ensemble is just an ensemble, in which all base learners are various autoencoder models. When all autoencoders in an ensemble are independent, that means the result of one model is not affected by that of the other, the ensemble is independent. With an independent autoencoder ensemble for anomaly detection, base learners are expected to capture different parts of the patterns in a given dataset and thus obtain diverse reconstruction errors. Each point in the dataset is scored with a reconstruction error by an autoencoder model in the independent ensemble. Therefore, the final reconstruction error of each data point is aggregated from its entire scores gained from every autoencoder model in the ensemble in some way. The most commonly used techniques are to take the maximum or the average of the whole reconstruction errors of each point. A data point is defined as an outlier when its final reconstruction error surpasses a certain threshold, which can be manually decided or computed by different methods.

2.4 Evaluation of Anomaly Detection Methods

How to evaluate the result of an unsupervised data mining task is generally a difficult question. It is even harder for evaluation of anomaly detection models. As they are not partitioning but ranking the data [45], how to appropriately assess the outliers' rankings and scores is still problematic. We will briefly explain two common methods used for evaluating outlier detection models in the literature, although some limitations related to them still exist [46].

2.4.1 Precision at k

The first method is precision@ k , which indicates the proportion of points in the top k results ranked as outliers in a dataset is actually abnormal. For instance, a dataset contains 50 points and we expect to have 5 outliers included. Based on the predictions of a trained model, top $k = 5$ points are ranked as outliers, in which only 2 among them are actually abnormal. Then the precision@5 should be $2/5$. This approach is problematic because most datasets used for anomaly detection problems are naturally highly imbalanced, that have very few outliers and numerous inliers. It can frequently happen that precision@ k is equal to zero as these few actual outliers are not in the top k data objects ranked as outliers. Moreover, the value of precision@ k also depends on the parameter k , which needs to be set up appropriately for each dataset. Another aspect is that precision@ k measures the ranking quality concerning only top k results, not in relation to the whole dataset.

2.4.2 ROC curve and ROC AUC Analysis

The more advanced and widely used measure in evaluating anomaly detection models is based on receiver operating characteristic (ROC) curves. ROC curves indicate the trade-off between the true positive rate (TPR) and the false positive rate (FPR). For the outlier detection problem, a ROC curve visualizes the trade-off between the proportion at which the model can accurately recognize anomalous points against the proportion at which it mislabels normal points as outliers for various portions of the test dataset [21]. As it is hard to evaluate the plot, the resulting monotone curve is turned into a measure by computing the area under this curve (AUC) [46]. Thus, to evaluate the accuracy of a model, we can measure the area under the resulting monotone curve. It is allowed to display some results in a single plot and compare them numerically. If which ROC curve is closer to the diagonal line, then its corresponding model is less accurate. The diagonal line represents the random guessing, for which both true positive rate and false positive rate will grow simultaneously and therefore the area under the curve derived from a random result will approximately take up half of the space. The area under a certain ROC curve will completely fill the entire available space when the corresponding model of the ROC

curve is perfectly accurate. That means the model detects correctly all outliers first and only then ranks inliers. In this case, the ROC AUC value obtains the maximum at 1.0. Basically, we can consider the ROC AUC value as the probability that a pair of outlier and inlier randomly selected will be correctly sorted with the outlier ranked before the inlier [23]. Figure 2.2 shows how ROC curves are plotted in a diagram as well as a detail explain of related aspects. The ROC curves of three different models are illustrated in the plot, in which the model corresponds to the blue curve is the best one, whereas that of the orange curve is the worst.

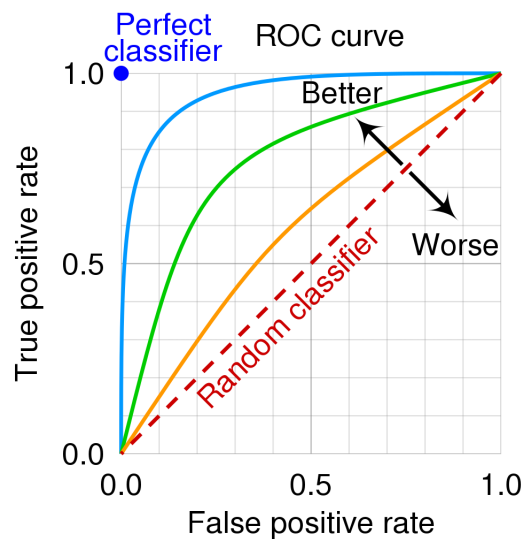


Figure 2.2: A diagram representing the ROC plot. [41]

In general, ROC curves and ROC AUC analysis have become the most popular method for evaluating the performance of anomaly detection models because of the ability to solve the class imbalance issue by using the relative frequencies. Moreover, it is really easy to visualize and compare the accuracy of one or several models as we can plot the ROC curves of different models trained on the same dataset on a single figure. From the plot, we can conclude about the performance of the models.

Chapter 3

Methodology

This chapter focuses on various variants of the sequential autoencoder ensemble implemented in this research. We will dig deep into their architectures as well as the approach to combining the scores from the base learners.

3.1 Sequential Autoencoder Ensemble

According to [2], anomaly detection ensembles can be approached in two ways, independent and sequential ensembles. Independent ensembles are to combine the strength of different independent base learners included together in order to obtain more robust models. On the other hand, sequential ensembles use the outputs of previous base learners to train the next model. It can mean training models to correct perceived errors of the previous ones [?]. Independent ensembles have recognized a more common exploration than sequential variants, in which RandNet [14] is one of the most remarkable architectures. However, to achieve a good performance, independent ensembles require a high variance between components, which ensures their base learners are able to capture different aspects of the underlying patterns [14] and support for each other to enhance the robustness of ensembles. Meanwhile, the former tend to produce similar base learners. As a result of the lack of diversity between base learners, their combination will not always outperform the best individual base learner [15]. To address the problem of variance, we suggest a sequential autoencoder ensemble, that can generate autoencoder models with highly different results as the prediction error of the previous model is used to correct the error of the next model. The final result of an ensemble is either a combination of that of the entire base learners, or only the last model [2]. The final prediction error of our sequential autoencoder ensemble is the average of that of the whole models included.

Figure 3.1 provides the architecture of the sequential autoencoder ensemble used in the research, that contains k autoencoder models as base learners. Given a dataset $D(n, m)$ with n data points and m features, on which the first base model is trained. Mean square

of difference between an input and its reconstruction is called reconstruction error, which will be added as a feature to the original dataset $D(n, m)$ before training the next model. This process iterates till k base learners are trained completely. In general, except for the first iteration using the dataset with m features, the remaining use the extended dataset with $m + 1$ features. The sequential autoencoder ensemble contains autoencoder models as base learners, which try to reconstruct their inputs. Therefore, we receive the output $D'(n, m)$, when fitting the original dataset $D(n, m)$ to the first model, and the output $D'(n, m + 1)$ by training the rest of base learners with the extended dataset $D(n, m + 1)$. The reconstruction error of each base detector is defined as below:

$$E_1 = MSE \left(D(n, m), D'(n, m) \right) \quad (3.1)$$

$$E_i = MSE \left(D(n, m + 1), D'(n, m + 1) \right) \quad \text{with } i = 2, \dots, k \quad (3.2)$$

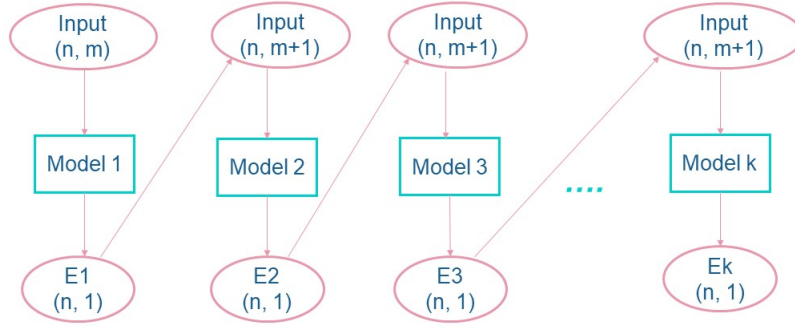


Figure 3.1: The architecture of sequential autoencoder ensemble used in the experiments

Algorithm 3.1 provides a detailed framework of our sequential autoencoder ensemble. The algorithm receives the training dataset $D(n, m)$, the testing dataset $T(l, m)$, and the number of base detectors k as parameters. O is initialized as an empty set, which will contain reconstruction errors of each base detector concerning the testing dataset. In each iteration, we first train an autoencoder using the training data. Then, the autoencoder tries to reconstruct the training data. By calculating mean square error between training data and its reconstruction, we obtain the reconstruction error E_i , which will be added to the original dataset $D(n, m)$ as a feature before assigning it to the training data used for the next iteration. Similarly, we use the autoencoder to reconstruct the test data. Its reconstruction, test reconstruction, is saved into the set O and is used as a feature added

to the original testing dataset $T(l, m)$ to generate the extended test data for the next iteration. After training k -base learners, by averaging the entire elements in the set O , we obtain the reconstruction error of the sequential autoencoder ensemble, which will be used to calculate the AUC score and is the base to identify outlier by applying a threshold on it.

Input: Dataset $D(n, m)$, Test set $T(l, m)$, number of base learners k

```

1: procedure SEQUENTIALENSEMBLE( $D(n, m), T(l, m), k$ )
2:    $O \leftarrow \{\}$ 
3:   training data  $\leftarrow D(n, m)$ 
4:   test data  $\leftarrow T(l, m)$ 
5:   for  $i \leftarrow 1, k$  do
6:     autoencoder  $\leftarrow$  training model(training data)
7:     training reconstruction  $\leftarrow$  autoencoder.predict(training data)
8:      $E_i \leftarrow$  mean square error(training data, training reconstruction)
9:     training data  $\leftarrow D(n, m) \cup E_i$ 
10:    test reconstruction  $\leftarrow$  autoencoder.predict(test data)
11:     $O_i \leftarrow$  mean square error(test data, test reconstruction)
12:     $O \leftarrow O \cup O_i$ 
13:    test data  $\leftarrow T(l, m) \cup O_i$ 
14:  end for
15:  reconstruction error of sequential ensemble  $\leftarrow$  the average of elements in  $O$ 
16:  return reconstruction error of sequential ensemble
17: end procedure

```

Algorithm 3.1: Sequential Autoencoder Ensemble Algorithm

3.2 Full Sequential Autoencoder Ensemble

From the idea of the sequential autoencoder ensemble proposed above, we develop another variant called full sequential autoencoder ensemble, in which all previous base learners support adjusting the next one. While the former uses the prediction error of the last trained detector as a feature added to the original dataset, the latter utilizes the prediction errors of all previously trained learners for the next iteration. Similar to the sequential autoencoder ensemble, the average of the entire base learners' prediction errors defines the final prediction error of the full sequential autoencoder ensemble.

Figure 3.2 illustrates the architecture of the full sequential autoencoder ensemble with k autoencoder models as base detectors. By training the first base model with a dataset $D(n, m)$, we figure out the reconstruction error E_1 , which will be assigned to $D(n, m)$

for the next iteration. After fitting the second base model with the extended dataset $D(n, m + 1)$, the reconstruction error E_2 computed will be added as a feature to the extended dataset $D(n, m + 1)$ used in the previous iteration instead of the original dataset $D(n, m)$. Principally, $D'(n, m)$ is the reconstructed output of the dataset $D(n, m)$ after the first iteration, whereas $D'(n, m + i - 1)$ is the output by training the base detector i with the extended dataset $D(n, m + i - 1)$. The reconstruction error of each base learner is defined below:

$$E_1 = \text{MSE} \left(D(n, m), D'(n, m) \right) \quad (3.3)$$

$$E_i = \text{MSE} \left(D(n, m + i - 1), D'(n, m + i - 1) \right) \quad \text{with } i = 2, \dots, k \quad (3.4)$$

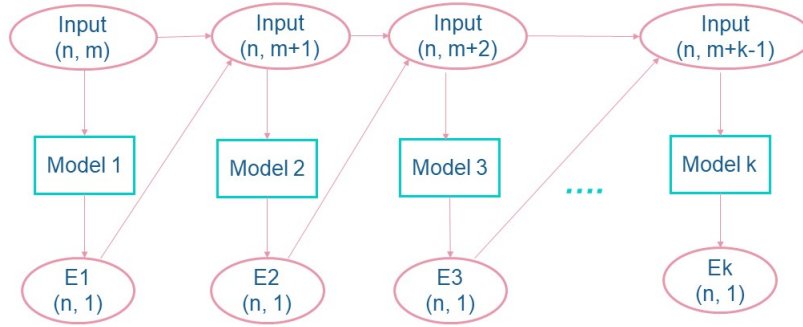


Figure 3.2: The architecture of full sequential autoencoder ensemble used in the experiments

The detailed framework of a full sequential autoencoder ensemble is demonstrated in the algorithm 3.2. This framework is quite similar to that of the sequential autoencoder ensemble except for some points. In line 9, instead of being added to the original dataset $D(n, m)$, the reconstruction error E_i is assigned to the training data of the current iteration as a feature to generate a new training dataset $D(n, m + i - 1)$ for the next base learner. The similarity is also seen in line 13, where the reconstruction error O_i is added to the test data to create a new test data $T(n, m + i - 1)$ for the next iteration. The reconstruction error of the full sequential autoencoder ensemble, which is the base to identify the AUC score of the model, is obtained by averaging the whole elements in the set O after training k -base detectors.

Input: Dataset $D(n, m)$, Test set $T(l, m)$, number of base learners k

```

1: procedure FULLSEQUENTIALENSEMBLE( $D(n, m), T(l, m), k$ )
2:    $O \leftarrow \{\}$ 
3:   training data  $\leftarrow D(n, m)$ 
4:   test data  $\leftarrow T(l, m)$ 
5:   for  $i \leftarrow 1, k$  do
6:     autoencoder  $\leftarrow$  training model(training data)
7:     training reconstruction  $\leftarrow$  autoencoder.predict(training data)
8:      $E_i \leftarrow$  mean square error(training data, training reconstruction)
9:     training data  $\leftarrow$  training data  $\cup E_i$ 
10:    test reconstruction  $\leftarrow$  autoencoder.predict(test data)
11:     $O_i \leftarrow$  mean square error(test data, test reconstruction)
12:     $O \leftarrow O \cup O_i$ 
13:    test data  $\leftarrow$  test data  $\cup O_i$ 
14:  end for
15:  reconstruction error of full sequential ensemble  $\leftarrow$  the average of elements in  $O$ 
16:  return reconstruction error of full sequential ensemble
17: end procedure

```

Algorithm 3.2: Full Sequential Autoencoder Ensemble Algorithm

3.3 Synchronizing Sequential Autoencoder Ensemble

With the approach that all previous base learners participate in correcting the next detector, we implement another variant called synchronizing sequential autoencoder ensemble. Instead of reshaping a base learner's prediction errors to get only one feature, we keep their dimensions intact as features that will be added to the input of the next iteration.

An architecture of synchronizing sequential autoencoder ensemble including k base learners is indicated in figure 3.3. After fitting the first base learner with a dataset $D(n, m)$, the prediction error S_1 computed by squaring the difference between the input and its reconstruction will be added as features to $D(n, m)$ before training the next model. Since the dimension of S_1 is kept the same as that of the input, the second base learner is trained with the extended dataset $D(n, m + m)$. Generally, the architecture looks quite similar to that of a full sequential autoencoder ensemble except for the method to calculate the prediction error of each base learner. By fitting the base learner i with the input $D(n, m * i)$, we obtain the reconstruction $D'(n, m * i)$. The reconstruction error of each base learner is defined below:

$$S_i = \left(D(n, m * i) - D'(n, m * i) \right)^2 \quad \text{with } i = 1, \dots, k \quad (3.5)$$

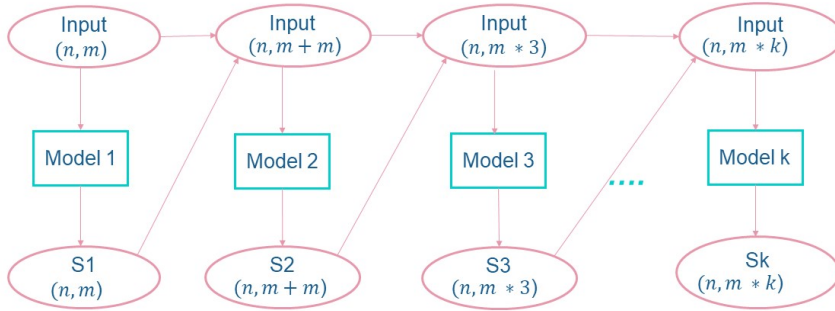


Figure 3.3: The architecture of synchronizing sequential autoencoder ensemble

Algorithm 3.3 presents the framework of a synchronizing sequential autoencoder ensemble containing k -base detectors. Except for lines 8 and 11, where the prediction errors of training data and test data are defined variously than that of a full sequential autoencoder ensemble, the remaining is quite similar.

Input: Dataset $D(n, m)$, Test set $T(l, m)$, number of base learners k

```

1: procedure SYNCHRONIZINGSEQUENTIALENSEMBLE( $D(n, m), T(l, m), k$ )
2:    $O \leftarrow \{\}$ 
3:   training data  $\leftarrow D(n, m)$ 
4:   test data  $\leftarrow T(l, m)$ 
5:   for  $i \leftarrow 1, k$  do
6:     autoencoder  $\leftarrow$  training model(training data)
7:     training reconstruction  $\leftarrow$  autoencoder.predict(training data)
8:      $S_i \leftarrow (\text{training data} - \text{training reconstruction})^2$ 
9:     training data  $\leftarrow$  training data  $\cup S_i$ 
10:    test reconstruction  $\leftarrow$  autoencoder.predict(test data)
11:     $O_i \leftarrow (\text{test data} - \text{test reconstruction})^2$ 
12:     $O \leftarrow O \cup O_i$ 
13:    test data  $\leftarrow$  test data  $\cup O_i$ 
14:  end for
15:  reconstruction error of synchronizing sequential ensemble  $\leftarrow$  the average of  $O$ 
16:  return reconstruction error of synchronizing sequential ensemble
17: end procedure

```

Algorithm 3.3: Synchronizing Sequential Autoencoder Ensemble Algorithm

Chapter 4

Experiments and Evaluation

This chapter presents the various experiments conducted in order to examine the performance of different approaches for anomaly detection. Besides implementing experiments on autoencoder models, we focus mainly on evaluating the performance of autoencoder ensemble variants.

4.1 Overview of datasets

Table 4.1 lists the entire datasets used for experiments in this research. All datasets were divided into subsets for training and testing. We used datasets with various sizes and dimensions to obtain a more comprehensive and robust evaluation of the performance of different model types. For instance, some used high-dimensional datasets like gas-drift and speech with 128 and 400 features respectively. In contrast, there are also a few low-dimensional datasets with only 5 features like phoneme and pollen.

Table 4.1: List of datasets used in experiments and their size

Dataset	Shape of training data	Shape of testing data	Number of anomalies in testing data
cardio	(1479, 21)	(352, 21)	176
gas-drift	(1796, 128)	(1538, 128)	769
satellite	(3080, 36)	(2638, 36)	1319
magic-telescope	(8633, 11)	(7398, 11)	3699
pendigits	(6558, 16)	(312, 16)	156
phoneme	(2673, 5)	(2290, 5)	1145
pollen	(1347, 5)	(1154, 5)	577
speech	(3564, 400)	(122, 400)	61
waveform	(1185, 40)	(1014, 40)	507

4.2 The Approach with Autoencoder Model

4.2.1 The Architecture of Autoencoder

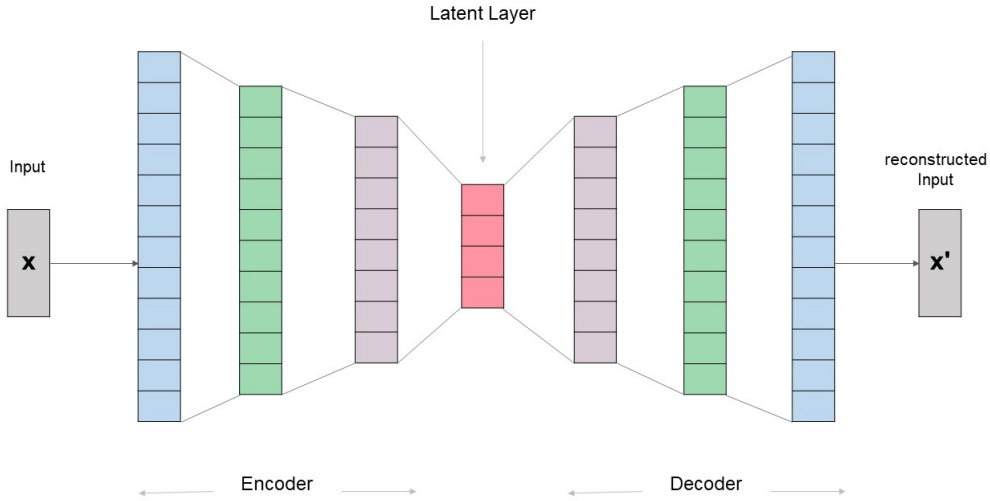


Figure 4.1: The architecture of Autoencoder used in the experiments

Figure 4.1 shows the architecture of autoencoder used in the experiments, inspired by [13]. The network includes seven layers - one input, two hidden in the encoder block, and the same is replicated in the decoder separated from the former by the latent layer in between. Under the condition that the number of hidden layers in the encoder and decoder blocks are equal, that in an autoencoder are not limited. After conducting experiments with some different quantities of hidden layers, we come up with the most suitable autoencoder architecture for our research as above. Having too many hidden layers does not mean the model becomes more accurate. Instead of learning the important lower-dimensional features representing data most, this model tries to copy the input to output mapping [13]. We keep the number of neurons n in the latent space as a variable that helps our experiments to be adjusted easier. The figure of neurons in the remaining layers tunes with the rate of $(n * 2)^i$ where $i \in \{1, 2, 3\}$ and comply with the autoencoder architecture. To support preventing overfitting, each hidden layer is set with a probability of eliminating a node in the neural network at 0.1 through the dropout layer. The non-linear activation function Relu is used in all layers except for the output layer, where the Sigmoid function is applied. In addition, we choose Adam (Adaptive moment estimation), a combination of the advantages of both AdaGrad and RMSProp optimization algorithms [31], as the optimization algorithm to update the weights in the backpropagation. After experimenting with different loss functions, the mean square error (MSE) has been indicated as the most appropriate loss function for our model as it efficiently detects outliers with large errors

by giving higher weight to them. We keep latent space as a variable that should be set up appropriately according to each used dataset. Values of some hyper-parameters used in autoencoder models are mentioned in table 4.2.

Table 4.2: Hyperparameters used for autoencoder models in experiments

Hyperparameter	Value
Drop Out Rate	0.1
Epoch Size	2000
Batch Size	64

4.2.2 Hyperparameters Analysis

The purpose of implementing experiments regarding some hyperparameters like batch size, epochs, and latent space is to analyze their effects on the performance of an autoencoder model for anomaly detection problems and figure out the most suitable values of hyperparameters for each dataset used in this thesis. The autoencoder architecture used in the experiments is introduced in Section 4.2.1, in which batch size, epoch size, and latent space are considered variables needed to initialize properly for the aim of each execution.

a. Batch size and Epoch size

Table 4.3: Latent size used to train autoencoder models for each dataset

Dataset	Latent size
cardio	21
gas-drift	112
satellite	192
magic-telescope	112
pendigits	16
phoneme	96
pollen	32
speech	112
waveform	112

To see how the batch size hyperparameter affects the performance of an autoencoder model, we trained multiple autoencoder models for each dataset with various batch sizes. The list of used batch sizes is $\{16, 32, 64, 96, 128, 256, 512\}$. All trained models are initialized with an epoch size of 2000 and respective latent spaces for each dataset listed in table 4.3. The initialization of epoch size and latent spaces like the above is based on some findings

during testing experiments and will be discussed later. Similarly, in order to inspect the influence of the epoch hyperparameter on an autoencoder model for anomaly detection, we kept the batch size at 64 and latent spaces for each dataset as in table 4.3. Then we trained autoencoder models for each dataset with different epoch sizes in $\{50, 100, 200, 500, 1000, 1500, 2000, 2500, 3000, 3500, 4000\}$.

The AUC score of each trained model concerning the adjustment of batch and epoch sizes are plotted in figures 4.2 and 4.3 respectively. As we can see in figure 4.2 the dependency between batch sizes and AUC scores of trained models is not high. In particular, most datasets tend to obtain a stable AUC score, when the batch size is increased. Only phoneme and speech datasets stay out of this tendency, but the fluctuation of their AUC scores is not significant. From the experiment, we found out that the batch size parameter can help to improve the performance of an autoencoder model for anomaly detection problems to some extent. Although each trained model concerning a given dataset can attain the highest AUC score at a specific value, we choose the most suitable batch size of 64 for all datasets in the remaining executions in this thesis for the aim of generalizing experiments. Figure 4.3 shows that the epoch parameter has a similar trend with the batch size. To improve the performance of an autoencoder model, we can consider to fine-tune the epoch parameter for a given dataset. However, the efficiency of this approach may not be desirable because the influence of the epoch parameter on the AUC score of a model indicated in the experiment is weak. Moreover, a concern regarding this parameter is computational cost, which can reach a high volume to finalize the best epoch size for a dataset. By executing the experiment with epoch sizes, we found the most appropriate epoch of 2000 for all datasets in the next experiments.

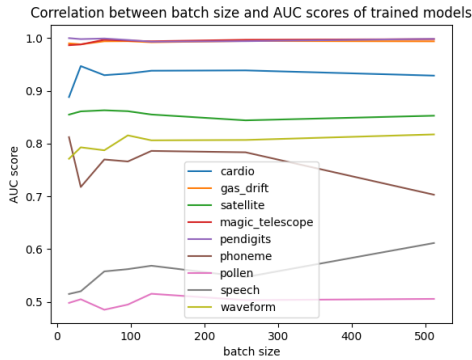


Figure 4.2: The influence of batch size

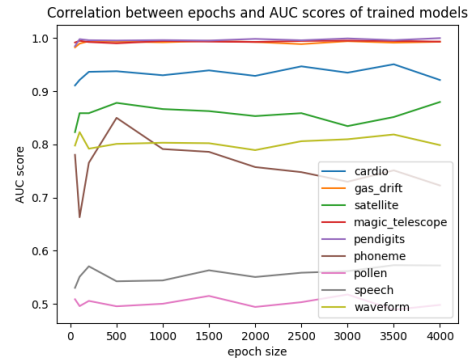


Figure 4.3: The influence of epoch size

b. Latent Space

Next, we focus on examining the influence of the latent space parameter on the performance of an autoencoder model for anomaly detection problems. For each dataset, we

trained multiple autoencoder models with different latent space values in $\{4, 8, 16, 32, 64, 96, 112, 128, 144, 160, 192, 256, 320\}$. The whole trained models were initialized with a batch size of 64 and an epoch size of 2000.

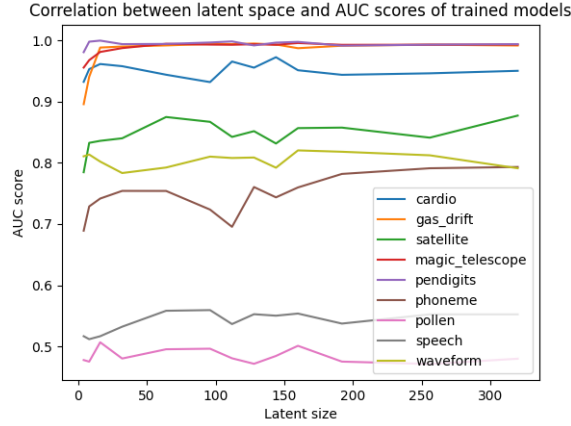


Figure 4.4: The influence of latent space on an autoencoder model

Figure 4.4 indicates the correlation between the latent space parameter and AUC scores of autoencoder models trained for anomaly detection problems. The experiment has proved that the latent space parameter affects significantly the performance of an autoencoder model. The plot in figure 4.4 shows a substantial fluctuation in AUC scores concerning latent sizes of autoencoder models, except for those of gas-drift, magic-telescope, and pendigits datasets. The possible reason for stability in the AUC scores of the mentioned datasets is that they are quite easy to train. We obtained really high AUC scores when training the models with these datasets, even though we have not done any fine-tuning methods yet. In general, by setting a suitable latent space, we can totally train an outperformed autoencoder model with only a shallow neuron network. However, it may cause a high computational cost to figure out the best latent space for a particular dataset as it seems that there are no fixed principles to support the finding process more effectively.

From experiments, we see that the latent space of an autoencoder does not need to be smaller than the number of features of a dataset. A dataset with few features can get the highest AUC score by using an autoencoder with a large latent space. For this point, we implemented an experiment for Phoneme and Pollen datasets containing only five features. We used the same autoencoder architecture as mentioned in 4.2.1 with a batch size of 64 and an epoch of 2000. Figure 4.5 shows ROC curves and AUC scores of both datasets fitted to the autoencoder model with a latent size of 4, whereas figure 4.6 indicates the case of using latent sizes of 320 and 16 for Phoneme and Pollen datasets respectively. By using an autoencoder model with a latent size of 4, we obtained lower AUC scores for both

datasets than using an autoencoder model with higher latent sizes of 320 for Phoneme and 16 for Pollen.

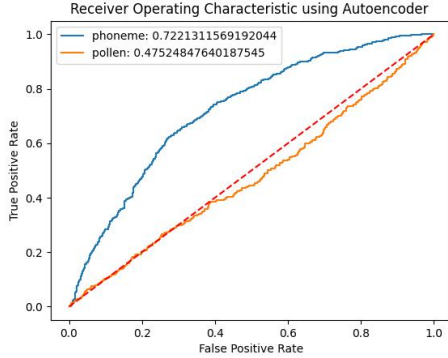


Figure 4.5: The ROC curves of datasets trained on autoencoder models with a latent space of 4

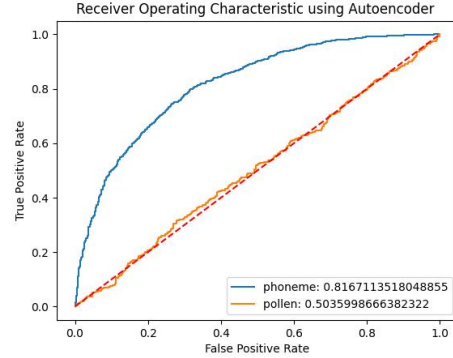
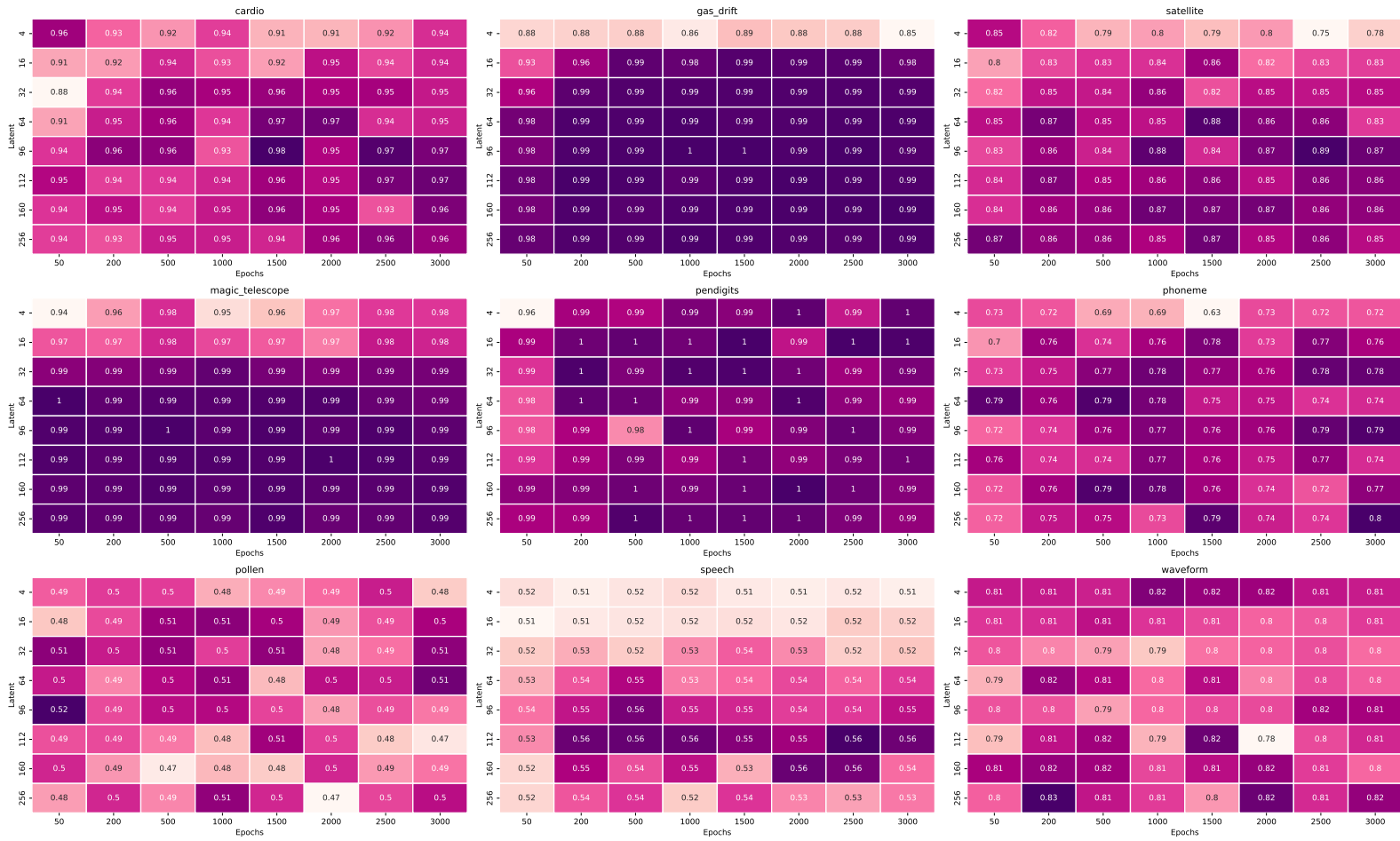


Figure 4.6: The ROC curves of Phoneme and Pollen datasets trained on autoencoder models with latent spaces of 320 and 16 respectively

Another interesting aspect we explored during implementing experiments is that latent space may have a positive correlation with epoch parameters. To inspect this point, we trained multiple autoencoder models for each dataset. In particular, each dataset was fitted to an autoencoder model initialized with an epoch in $\{50, 200, 500, 1000, 1500, 2000, 2500, 3000\}$, which was trained iteratively by using a latent space in $\{4, 16, 32, 64, 96, 112, 160, 256\}$ each time. AUC scores of each dataset according to specific epoch and latent values are plotted in a heat map. Figure 4.7 summarizes heat map plots of all datasets. We can see that if we choose a couple of latent space and epoch parameters, in which the former is really small and the latter is too big, or vice versa, then most models tend to obtain a lower AUC score. The highest AUC scores of each dataset in figure 4.7 concentrate mainly on the middle, bottom right, or top left of the plot, where the distance between latent space and epoch values is not too big. Of course, to prove the accuracy of the statement, we need to do more experiments, which are supposed to reach out of this thesis. However, this exploration can be considered an effective suggestion to find the suitable couple of latent space and epoch values for autoencoder models of each dataset.

Figure 4.7: Correlation between latent space and epoch size



c. Conclusion about the influence of hyperparameters

In general, all batch, epoch, and latent space parameters contribute to improving an autoencoder model's performance. Figure 4.8 illustrates the influence extent of each parameter on an autoencoder model. To attain the plot, we computed firstly the standard deviation of AUC scores of each dataset according to each result obtained by executing experiments concerning hyperparameters as mentioned in 4.2.2.a and 4.2.2.b. Then, we applied the bar plot function in the seaborn library to represent the standard deviation of each variable. The plot shows that the latent space parameter has the highest influence on an autoencoder model's performance, whereas the effect of the epoch parameter is the weakest. With a set of proper hyperparameters, we can totally attain a simple but robust autoencoder model for anomaly detection problems.

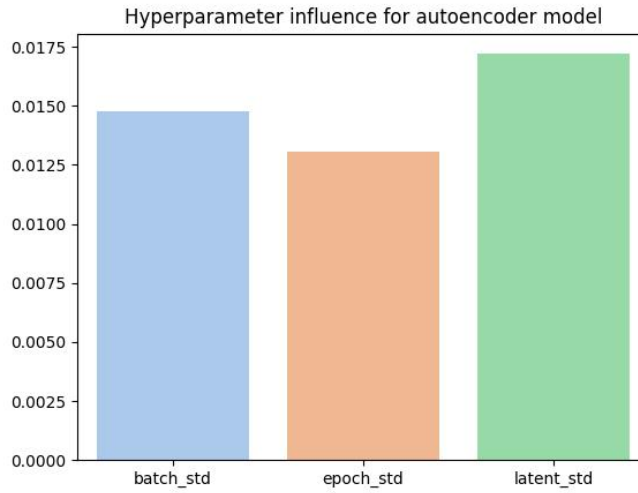


Figure 4.8: The influence of hyperparameters on an autoencoder model

4.2.3 Performance of Autoencoder Model

After finding the most suitable set of hyperparameters for each dataset we trained autoencoder models to detect anomalies. The architecture of trained autoencoder models is introduced in section 4.2.1. They are initialized with a batch size of 64 and an epoch of 2000. The latent space of each model for each dataset is listed respectively in table 4.3. Figure 4.9 indicates the ROC curves and AUC scores of various datasets trained with the autoencoder models. As we can see, most autoencoder models performed very well except for those using Pollen and Speech datasets, which are actually difficult to train. To have these excellent autoencoder models, we implemented various experiments as mentioned in section 4.2.2 with the aim of finding the most proper hyperparameters for each dataset. Overall, an autoencoder model containing only a few hidden layers can perform very well

at detecting outliers, when it is initialized with suitable hyperparameters. Nevertheless, searching for appropriate hyperparameters may lead to a high computational cost and be time-consuming. Moreover, this approach is not effective in general since it does not generalize about all anomaly detection problems. That means we need to find a proper hyperparameter set for a particular dataset to be able to train a well-performing autoencoder model.

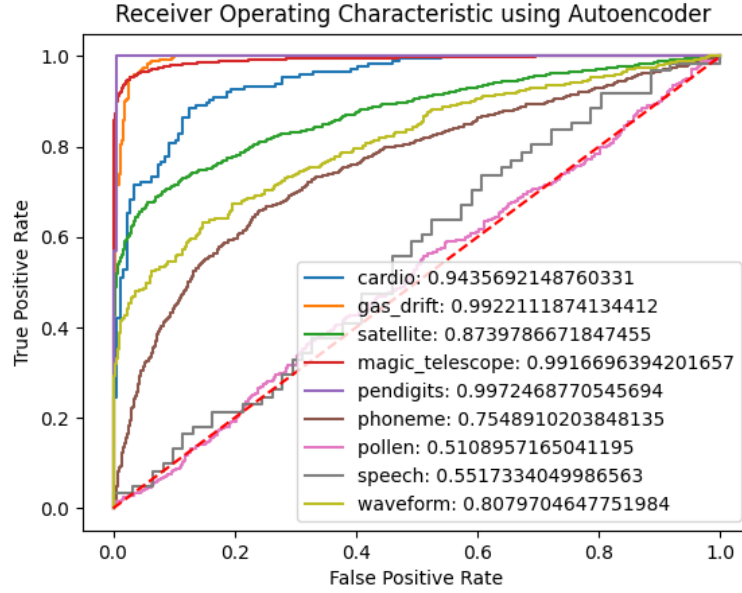


Figure 4.9: The ROC curves of various datasets trained with autoencoder models

4.3 The Approach with Autoencoder Ensemble Models

In this part, we focus on the experiments with various types of autoencoder ensembles. The architecture of the base models in each ensemble is introduced in section 4.2.1. Each type of base learner is initialized with a set of latent, batch, and epoch size parameters, which are listed in tables 4.4 and 4.5. To gain a more general overview of the performance of different models, we also tried to train our datasets with a RandNet model [14] by using the code from [1].

Table 4.4: Hyperparameter values initialized for each type of base learners

Hyperparameter	Learner 1	Learner 2	Learner 3	Learner 4	Learner 5
Batch size	64	64	64	100	64
Epoch size	2000	1000	1000	300	500

Table 4.5: Latent size used to train each type of base learners for each dataset

Dataset	Learner 1	Learner 2	Learner 3	Learner 4	Learner 5
cardio	21	4	4	4	32
gas-drift	112	32	16	16	64
satellite	192	4	16	16	32
magic-telescope	112	4	4	4	8
pendigits	16	4	4	4	8
phoneme	96	4	4	4	4
pollen	32	4	4	4	4
speech	112	64	256	256	128
waveform	112	32	32	32	64

4.3.1 Performance of Independent Autoencoder Ensembles

RandNet is an independent ensemble method in which some of the connections between layers in each autoencoder included are randomly dropped [14]. The RandNet architecture from [1] contains 200 autoencoder submodels. Each submodel is initialized with an epoch of 300 and a batch size of 100. The latent space of each autoencoder submodel is adjusted based on the parameter α and the number of features of the dataset used. More details about the RandNet model used for our next experiment can be found at [1]. By fitting all our datasets in the RandNet model, we received the AUC score for each dataset listed in table 4.6. Besides, we generated an *independent autoencoder ensemble* (shortcut as **IAE**) architecture for experiments. Our independent ensemble contains n ($n \geq 2$) different autoencoders as base learners. The structure of each autoencoder included is introduced in section 4.2.1. To ensure that all base learners are diverse, we set a different random seed value for each autoencoder used. The final prediction errors of a trained independent ensemble is the average of all prediction errors of its base detectors.

Firstly, we trained two different autoencoder models for each dataset. The unoptimized autoencoder is executed by using learner 4, while the optimized autoencoder uses learner 1 and is also the model mentioned in section 4.2.3. Next, two various independent ensemble models are trained with the unoptimized and optimized autoencoders as base learners separately. Both the unoptimized and optimized IAEs contain 60 base learners. The AUC scores of each dataset fitted to different types of models are shown in table 4.6. The optimized autoencoder indicates a better result than the unoptimized as the former's parameters were fine-tuned very well, whereas those of the latter were randomly selected. However, the performance of both autoencoder models exceeds the RandNet, an independent ensemble containing 200 submodels. The first reason for this situation may come from the improper latent sizes initialized for each submodel in the RandNet since latent space

has a strong effect on the performance of an autoencoder. In addition, the RandNet is designed by dropping randomly some connections between layers in each autoencoder included. It may happen that the removed connections bring the most important information that should be learned. As a result, the RandNet may learn only from weak connections with little information and hence, perform worse than its base models. Another point is that dropping randomly some connections between layers in each autoencoder may generate submodels with low variances. Therefore, the components in RandNet cannot capture various aspects of underlying patterns in data [14]. An ensemble with a low diversity between base learners may perform worse than the best individual base learner [15]. In our experiment, the RandNet’s performance cannot surpass the unoptimized autoencoder using the same epoch and batch size. Meanwhile, our independent ensemble models are constructed with an assurance about the diversity between base learners by setting a different random seed value for each component. As we can see in table 4.6, both independent ensembles obtained better performances than their components. Overall, whether an independent autoencoder ensemble may outperform its base learners depends mainly on the variance between the latter. Components with high variances can learn different aspects of the patterns in data and hence boost the robustness of their ensemble.

Table 4.6: Comparison AUC scores of autoencoder models, independent autoencoder ensembles, and RandNet. The best AUC is highlighted in boldface

Dataset	Unoptimized Autoencoder	Unoptimized IAE	Optimized Autoencoder	Optimized IAE	RandNet
cardio	0.9127	0.9311	0.9436	0.9527	0.9033
gas-drift	0.9787	0.9771	0.9922	0.9935	0.9953
satellite	0.8297	0.8311	0.8740	0.8700	0.7974
magic-telescope	0.9730	0.9766	0.9917	0.9967	0.9598
pendigits	0.9910	0.9921	0.9972	0.9997	0.9675
phoneme	0.7605	0.7250	0.7549	0.8353	0.6821
pollen	0.4940	0.4922	0.5109	0.5036	0.4938
speech	0.5528	0.5539	0.5517	0.5493	0.5751
waveform	0.7883	0.8095	0.8080	0.8258	0.8273
Average	0.8090	0.8098	0.8249	0.8363	0.8002

4.3.2 Performance of Sequential Autoencoder Ensembles

In this part, our experiments focus on studies concerning the *sequential autoencoder ensemble* architecture mentioned in section 3.1. A sequential autoencoder ensemble (shortcut as **SAE**) is expected to generate autoencoder models with high variances as the reconstruction error of the previous model is used to correct the error of the next one. That means the

reconstruction error of the last trained autoencoder model is considered a feature added to the original dataset before fitting it to the next learner. The diversity between base learners in a sequential ensemble allows its components to learn various aspects of data and hence enhance the robustness of the ensemble. Therefore, a sequential autoencoder ensemble is supposed to perform more excellently than its base learners.

To examine the performance of the sequential autoencoder ensemble architecture, we trained first a sequential ensemble containing 60 optimized autoencoders mentioned in section 4.2.3 as base learners. Table 4.7 gives an overview of the AUC scores of all datasets fitted to different models. The optimized IAE, as explained in section 4.3.1, contains 60 optimized autoencoders as base learners. In almost all datasets, the optimized SAE outperforms the other models. Although the optimized IAE has a higher average of AUC scores than that of the optimized autoencoder, it is still lower than the optimized SAE.

Table 4.7: Comparison AUC scores of autoencoder model, independent autoencoder ensemble and sequential autoencoder ensemble. The best AUC is highlighted in boldface.

Dataset	Optimized Autoencoder	Optimized IAE	Optimized SAE
cardio	0.9436	0.9527	0.9550
gas-drift	0.9922	0.9935	0.9937
satellite	0.8740	0.8700	0.8750
magic-telescope	0.9917	0.9967	0.9970
pendigits	0.9972	0.9997	0.9998
phoneme	0.7549	0.8353	0.8418
pollen	0.5109	0.5036	0.5165
speech	0.5517	0.5493	0.5598
waveform	0.8080	0.8258	0.8238
Average	0.8249	0.8363	0.8403

Since each model in a sequential autoencoder ensemble is trained to learn from the reconstruction error of its predecessor and then correct itself, a SAE is supposed to outperform its components in most cases. To investigate this point, we trained five different sequential autoencoder ensembles containing 60 base models. Each ensemble uses a type of base learner listed in tables 4.4 and 4.5. For this experiment, we will use independent autoencoder ensembles as competitors. Therefore, we trained similarly five independent autoencoder ensembles of 60 components. Besides, each type of learner listed in tables 4.4 and 4.5 is considered an autoencoder and trained for comparison. The averaging AUC scores of all datasets fitted into multiple autoencoders, independent autoencoder ensembles, and sequential autoencoder ensembles are plotted in figure 4.10. As we can see, all

sequential autoencoder ensembles perform much better than their base learners, the autoencoders. More interestingly, they also exceed their competitors in all the experiments.

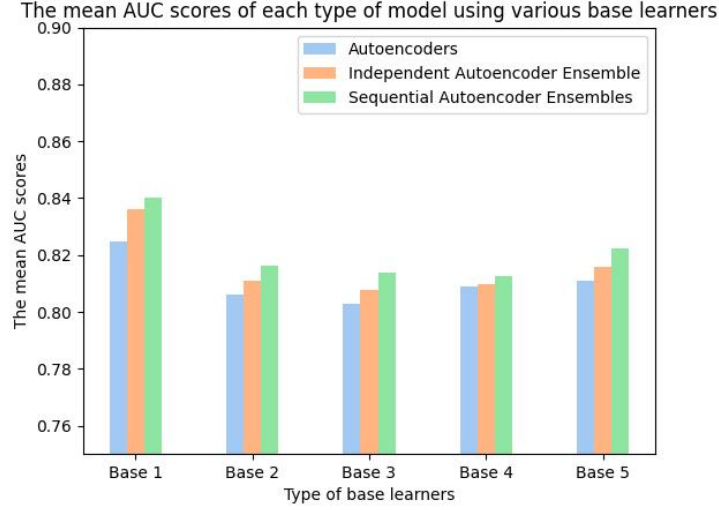


Figure 4.10: Comparison the average of AUC scores of autoencoder model, independent autoencoder ensemble and sequential autoencoder ensemble using different type of base learners

In order to inspect how the number of base learners affects the AUC performance of a sequential ensemble, we train an ensemble for each number of models from 2 to 200. The base learners used for this experiment apply parameters of the learner 1 mentioned in tables 4.4 and 4.5. The AUC performances for the *phoneme*, *pollen*, and *speech* datasets are shown in figure 4.11. In general, increasing the number of components in a sequential ensemble improves the performance of that ensemble. However, the more submodels in an ensemble do not mean the higher performance of that ensemble. Figure 4.11 shows the performance of sequential ensembles including 200 submodels is even lower than that of models with fewer than 20 components. Meanwhile, the number of base learners in a sequential ensemble increases proportionally with computational costs.

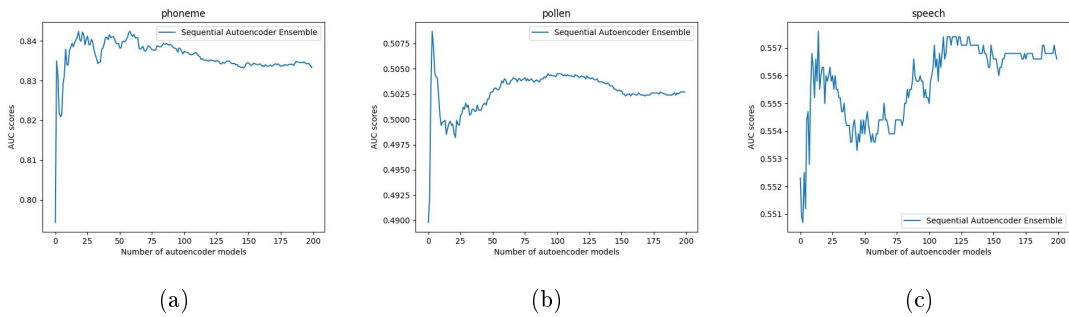


Figure 4.11: ROC AUC performance of each dataset relates to the number of base learners in a sequential autoencoder ensemble

4.3.3 Performance of Full Sequential Autoencoder Ensembles

A variant of sequential autoencoder ensemble, which is called *full sequential autoencoder ensemble* (shortcut as **FSAE**), is introduced in section 3.2. Instead of using only the prediction error of the last trained learner as a feature added to an original dataset, a full sequential autoencoder ensemble uses that of all previously trained learners for the next execution. In this part, we implement experiments to evaluate the performance of a full sequential autoencoder ensemble for anomaly detection.

Firstly, we train a full sequential autoencoder ensemble containing 60 optimized autoencoders mentioned in section 4.2.3 as base detectors. Table 4.8 is an extended version of table 4.7, in which the optimized FSAE’s performance for each dataset is completed. Although the average AUC score of the optimized FSAE is higher than that of the optimized SAE, it is not significant. Both of them seems to behave similarly and outperform the optimized IAE as well as their base learners, the optimized autoencoder.

Table 4.8: Comparison AUC scores of autoencoder model, independent autoencoder ensemble, sequential autoencoder ensemble, and full sequential autoencoder ensemble. The best AUC is highlighted in boldface.

Dataset	Optimized Autoencoder	Optimized IAE	Optimized SAE	Optimized FSAE
cardio	0.9436	0.9527	0.9550	0.9572
gas-drift	0.9922	0.9935	0.9937	0.9934
satellite	0.8740	0.8700	0.8750	0.8749
magic-telescope	0.9917	0.9967	0.9970	0.9971
pendigits	0.9972	0.9997	0.9998	0.9998
phoneme	0.7549	0.8353	0.8418	0.8385
pollen	0.5109	0.5036	0.5165	0.5165
speech	0.5517	0.5493	0.5598	0.5671
waveform	0.8080	0.8258	0.8238	0.8216
Average	0.8249	0.8363	0.8403	0.8407

Next, we train five various full sequential autoencoder ensembles containing 60 base models that use five different types of learners listed in tables 4.4 and 4.5. The average AUC scores of five full sequential autoencoder ensembles trained with multiple datasets are shown in figure 4.12, where that of other models are explained in section 4.3.2. Figure 4.12 indicates that full sequential autoencoder ensembles behave similarly to sequential autoencoder ensembles, even though some FSAEs perform better than SAEs. The performances of all independent autoencoder ensembles still cannot exceed that of FSAEs. Moreover, both FSAEs and SAEs outperform significantly their base learners.

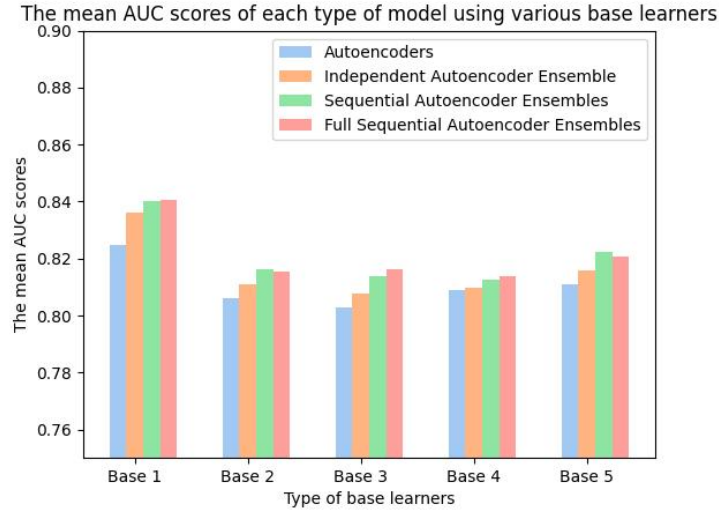


Figure 4.12: Comparison the average of AUC scores of autoencoder model, independent autoencoder ensemble, sequential autoencoder ensemble and full sequential autoencoder ensemble using different type of base learners

To examine the influence of the number of base learners in a full sequential autoencoder ensemble on its performance, we also train a FSAE for each number of models from 2 to 200, in which the learner 1 mentioned in tables 4.4 and 4.5 is used as base learners. Figure 4.13 is extended from figure 4.12 with the performance of FSAEs trained for the *phoneme*, *pollen*, and *speech* datasets are added. Similar to a sequential autoencoder ensemble, the number of components in a full sequential autoencoder affects the AUC score of a dataset substantially. The performances of FSAE and SAE reach the convergence at some point, where the former keeps stable or starts to decrease when the number of base learners in the ensemble increases. Therefore, to achieve a robust ensemble at low computational cost, optimizing the number of components in an ensemble generally is vital.

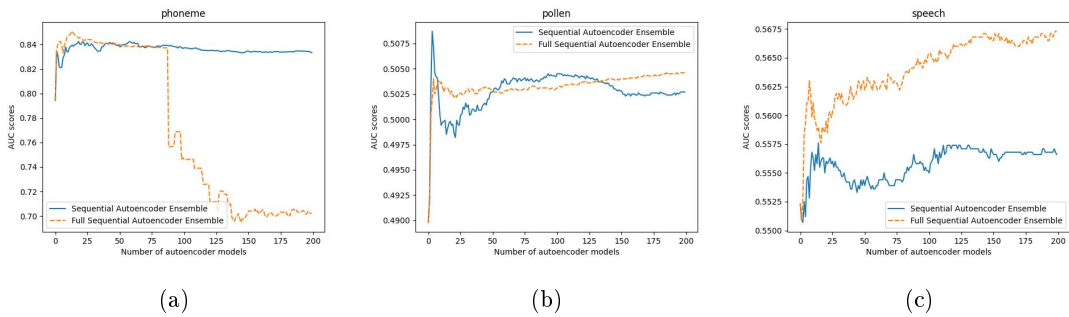


Figure 4.13: ROC AUC performance of each dataset relates to the number of base learners in sequential autoencoder ensemble and full sequential autoencoder ensemble

4.3.4 Performance of Synchronizing Sequential Autoencoder Ensembles

Another variant of sequential autoencoder ensemble proposed in this study is *Synchronizing Sequential Autoencoder Ensemble* (shortcut as **SSAE**). The difference from a full sequential autoencoder ensemble is that in a synchronizing sequential autoencoder ensemble, instead of reshaping a base learner’s reconstruction errors to get only one feature, we keep their dimensions intact as features that will be added to the input of the next execution.

To evaluate the performance, we train at first a synchronizing sequential autoencoder ensemble containing 60 optimized autoencoders mentioned in section 4.2.3 as base learners. However, as the number of features of each dataset fitted into each base learner will increase dramatically after every iteration, we receive an error related to the system’s limited computational capacity. Finally, we train only an SSAE with 5 optimized autoencoders as components. Table 4.9 is expanded from table 4.8, in which the optimized SSAE contains only 5 components, whereas all optimized IAE, SAE, and FSAE include 60 base learners. The results of the RandNet ensemble explained in section 4.3.1 are also included in table 4.9 for comparison. Even though this seems like an unfair comparison for the optimized SSAE, it performs quite well in this experiment. Its average AUC score exceeds not only the optimized autoencoder but also the RandNet.

Table 4.9: Comparison AUC scores of autoencoder model, independent autoencoder ensemble, sequential autoencoder ensemble, full sequential autoencoder ensemble, synchronizing sequential autoencoder ensemble, and RandNet. The best AUC is highlighted in boldface.

Dataset	Optimized Autoencoder	Optimized IAE	Optimized SAE	Optimized FSAE	Optimized SSAE	RandNet
cardio	0.9436	0.9527	0.9550	0.9572	0.9454	0.9033
gas-drift	0.9922	0.9935	0.9937	0.9934	0.9948	0.9953
satellite	0.8740	0.8700	0.8750	0.8749	0.8693	0.7974
magic-telescope	0.9917	0.9967	0.9970	0.9971	0.9958	0.9598
pendigits	0.9972	0.9997	0.9998	0.9998	0.9984	0.9675
phoneme	0.7549	0.8353	0.8418	0.8385	0.8037	0.6821
pollen	0.5109	0.5036	0.5165	0.5165	0.5146	0.4938
speech	0.5517	0.5493	0.5598	0.5671	0.5684	0.5751
waveform	0.8080	0.8258	0.8238	0.8216	0.8101	0.8273
Average	0.8249	0.8363	0.8403	0.8407	0.8334	0.8002

We want to have a glance at the influence of the number of base learners in an SSAE on the AUC performance. Thus, we plot the average AUC scores of all datasets trained with the above SSAE concerning each number of models from 1 to 5. For the purpose of comparison, we restrict ourselves to showing only results of the first 5 in 60 components of

the IAE, SAE, and FSAE as mentioned above. Figure 4.14 demonstrates the average AUC scores of all datasets related to each number of base learners in an ensemble. We see that the average AUC scores of IAE, SAE, and FSAE keep increasing when the number of base learners rises. But, the most interesting property is that after increasing continuously in the first three base learners, the AUC performance of the SSAE drops dramatically. One of the reasons for this decreasing AUC performance may result from the unchanged latent size of the base learners, which affects significantly the performance of an autoencoder as explained in section 4.2.2. The amount of features in every dataset increases multiple after each iteration, whereas the latent size remains stable. Consequently, the initialized latent size of each autoencoder in the SSAE may become unsuitable with the dataset fitting into it in later iterations. Therefore, we may need to adjust the latent size reasonably for each component in an SSAE, instead of keeping it unchanged as we proposed.

Because there are some limitations in the computational capacity of our system and several adjustments in the algorithm of an SSAE may be required to obtain a more robust ensemble, we will not go further to study this kind of ensemble in our thesis. However, the synchronizing sequential autoencoder ensemble architecture proposed could be potential and valuable for future works, as it seems to learn very fast and perform well after the first few base learners.

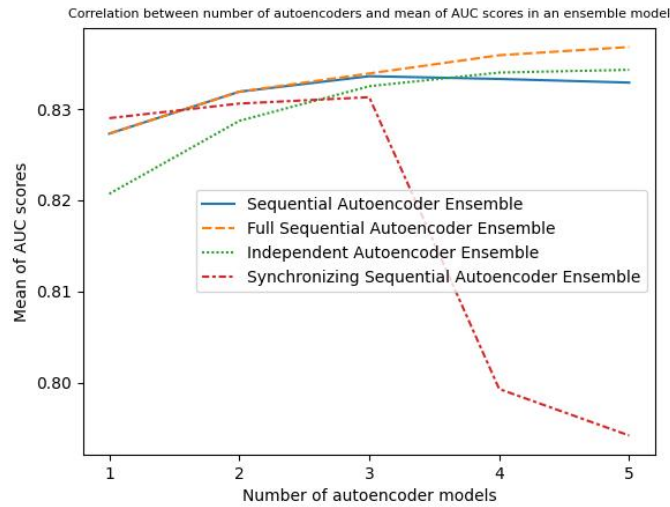


Figure 4.14: Comparison the average of AUC scores of autoencoder model, independent autoencoder ensemble, sequential autoencoder ensemble, full sequential autoencoder ensemble, and synchronizing sequential autoencoder ensemble using different type of base learners

4.4 Models Evaluation

Figure 4.12 shows that the performances of sequential autoencoder and full sequential autoencoder ensembles outperform those of independent autoencoder ensembles. However,

it still argues about the statistically significant difference between them. Therefore, we use a Wilcoxon test [42] to determine whether the performances of a sequential autoencoder and full sequential autoencoder ensembles are significantly different than an independent autoencoder ensemble. The Wilcoxon test is used to test the significance of the differences of the means in two dependent groups [42]. In our case, we execute Wilcoxon tests with the null hypothesis is that the differences in the performances of the two models are significant, and the alternative is that they are not significant. The null hypothesis is not rejected if the p-value is less than a multiple level of significance α [27], which we set at 1%.

In sections 4.3.2 and 4.3.3, we introduce the experiments to train five different sequential, independent, and full sequential autoencoder ensembles using five various types of base learners from 1 to 5. To perform Wilcoxon tests, we save the AUC scores of each dataset listed in 4.1, which we obtained by fitting them into five respective sequential autoencoder ensembles, into an array. We do the same for the cases of independent and full sequential autoencoder ensembles. Then, we implement the Wilcoxon tests of sequential and full sequential autoencoder ensembles against independent autoencoder ensemble as well as sequential autoencoder ensemble against full sequential autoencoder ensemble. In addition, we also perform Wilcoxon tests of autoencoder against independent, sequential, and full sequential autoencoder ensembles. The AUC scores of each dataset attained by training them with five various autoencoders listed in tables 4.4 and 4.5 are saved into an array. After that, we also implement necessary Wilcoxon tests. The p-value of each test is listed in table 4.10. Overall, all tests indicate that differences in the performances of the two models are statistically significant at $p < 0.01$, except for the tests of SAE again FSAE, and AE again IAE. Whereas both SAE and FSAE perform significantly better than IAE, the differences in their performances are not significant. This implies that both proposed sequential and full sequential autoencoder ensembles perform better than the independent autoencoder ensemble in general. Moreover, a sequential autoencoder ensemble behaves similarly to a full sequential autoencoder ensemble. The former can perform better than the latter in several datasets and vice versa. Although an independent autoencoder ensemble performs better than their base learners, the improvement is not significant. Meanwhile, we witness significant improvements of sequential and full sequential autoencoder ensembles in comparison with their base learners.

Table 4.10: The p-values obtained by implementing the Wilcoxon tests of sequential and full sequential autoencoder ensembles against independent autoencoder ensemble, the Wilcoxon test of sequential autoencoder ensemble against full sequential and the Wilcoxon tests of autoencoders against different ensembles

Test	p-value
SAE against IAE	4.917420710626175e-07
FSAE against IAE	4.335469782290602e-06
SAE against FSAE	0.6185166838524876
AE against IAE	0.022954077835947828
AE against SAE	5.778346121587674e-07
AE against FSAE	2.6416435616738454e-06

Finally, we want to evaluate the correlation between the number of autoencoders in an ensemble and AUC scores. Continuing with experiments mentioned in sections 4.3.2 and 4.3.3, we fit every dataset listed in table 4.1 into SAE, FSAE, and IAE models which are trained for each number of components from 2 to 200. Figure 4.15 shows the AUC scores of every dataset in correlation with the number of base learners in sequential, full sequential, and independent autoencoder ensembles. While the subplots from a to i demonstrate the AUC scores of each dataset separately, the last subplot summarizes the average AUC scores of all datasets related to each number of components in an ensemble. As we can see from all subplots represented for every dataset as well as the last subplot, the performances of ensembles reach the convergence at some point before starting to decrease or remaining tiny improvements. These convergences of almost all datasets are achieved with fewer than the first 25 submodels in an ensemble. The last subplot shows that the average correlation between the number of components and AUC scores seems approximately constant from some point for the sequential and independent autoencoder ensembles. This is reasonable because there is no interaction between base learners in an independent autoencoder ensemble. For a sequential autoencoder ensemble, a current component learns from the predecessor's reconstruction error which is adjusted by learning from the previous submodel to correct itself. Accumulatively, reconstruction errors are learned and adjusted to a minimum, then at some point, the next submodel has nothing to learn from its predecessor and hence keeps stable. We can see a similarity for the full sequential autoencoder ensemble to some extent, when reconstruction errors are completely learned and adjusted to a minimum, the next components have nothing to learn and thus remain stable. This aspect can be seen quite clearly from around the 30th to the 85th submodels of the full sequential autoencoder ensemble demonstrated in the last subplot. However, a component in a full sequential autoencoder ensemble learns not just from the last previous one, but from all previous submodels to adjust itself. Therefore, a full sequential autoencoder ensemble

tends to learn and obtain the best performance much faster than the other ensembles. That's why it can achieve the highest average AUC score after the first few submodels as displayed in the last subplot. As the number of components in a full sequential autoencoder ensemble continues to increase, the number of features of the dataset fitted to the next iteration also rises proportionally, while the latent size of each autoencoder in the ensemble is unchanged. It leads to the base learners in later iterations starting to perform poorer than their predecessors, when their latent sizes become unsuitable with the dataset fitted to them. Thus, we witness a rapid decrease in the average AUC score in the full sequential autoencoder ensemble from about the 86th submodel.

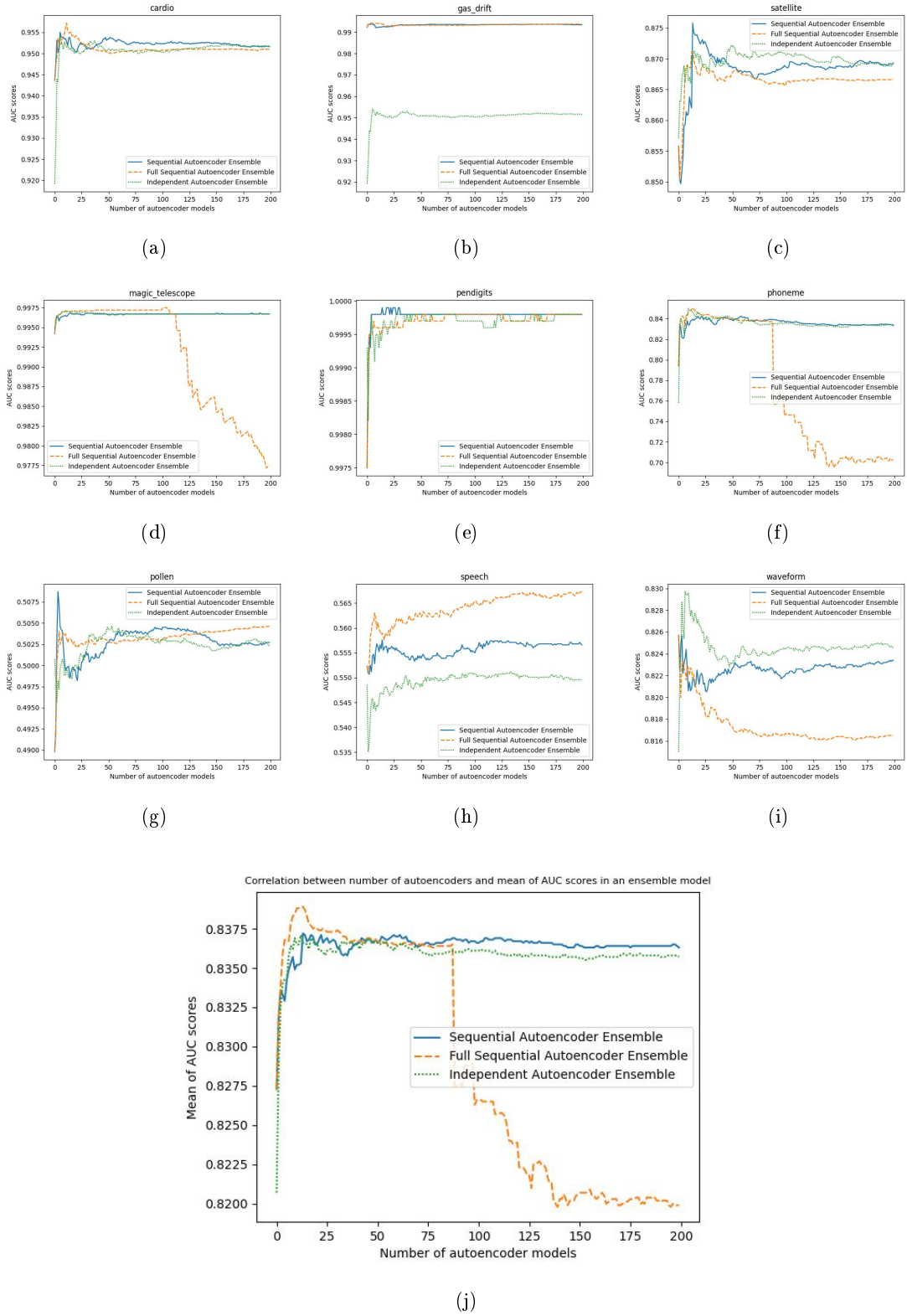


Figure 4.15: ROC AUC performance of each dataset relates to the number of base learners in sequential autoencoder ensemble, full sequential autoencoder ensemble, and independent autoencoder ensemble

Chapter 5

Conclusion and future work

In general, an autoencoder model containing only a few hidden layers can perform very well for anomaly detection tasks, when it is initialized with suitable hyperparameters, especially the latent space, which has the most influence on the performance of an autoencoder. However, searching for appropriate hyperparameters may not be an effective approach in general as it does not generalize about all anomaly detection problems. By combining multiple autoencoders with the same architecture, but different random seed values, we attain independent autoencoder ensembles, which perform better than their base learners in our experiments. But, their improvement seems to be not significant in comparison with their base learners. Because their components still tend to be similar to each other as they share the same architecture and hyperparameters' values. Both sequential and full sequential autoencoder ensemble techniques indicate their performances exceed that of independent autoencoder ensembles significantly. In addition, they behave quite similarly to each other. However, a full sequential autoencoder ensemble seems to learn faster than a sequential autoencoder since the former learns not just from the last previous submodel, but from all previous iterations to correct itself. This property of a full sequential autoencoder ensemble is an advantage, but also a disadvantage. As their submodels increase, the number of features of the dataset fitted to the next submodel also rises proportionally, whereas the latent size of each autoencoder remains intact. This triggers the poor performance of the base learners in later iterations, because the latent size becomes inappropriate with the dataset fitted to them. This aspect can be seen clearly in our experiments related to synchronizing sequential autoencoder ensemble technique.

Full and synchronizing sequential autoencoder ensemble techniques could be potential and valuable for future work since they seem to learn very fast and perform well after the first few submodels in our experiments. By adjusting the latent size of each autoencoder to become adaptive with each dataset fitted into the model, we may expect some improvements in the performance of full and synchronizing sequential autoencoder ensembles. Moreover, further studies about tuning hyperparameters of an ensemble could be essential as we have

seen how they affect an autoencoder’s performance in section 4.2.2. Besides tuning some parameters like latent space, epoch, and batch size, it could be a promised aspect to dig deeper into tuning loss function. While implementing experiments to figure out the most suitable autoencoder architecture for our thesis, we realized that the loss function plays a crucial role in the efficiency of a model. Just by using a different loss function, the performance of our models has changed a lot. Another aspect is the number of components in an ensemble. We have seen that increasing the number of submodels in an ensemble improves its performance. However, the more base learners in an ensemble does not mean the better it performs. Meanwhile, it leads to higher computational costs. Therefore, it will be worthy to research further about the optimization of the number of components in an ensemble.

List of Figures

2.1	An example of anomalies in a two-dimensional dataset	4
2.2	A diagram representing the ROC plot. [41]	14
3.1	The architecture of sequential autoencoder ensemble used in the experiments	16
3.2	The architecture of full sequential autoencoder ensemble used in the experiments	18
3.3	The architecture of synchronizing sequential autoencoder ensemble	20
4.1	The architecture of Autoencoder used in the experiments	22
4.2	The influence of batch size	24
4.3	The influence of epoch size	24
4.4	The influence of latent space on an autoencoder model	25
4.5	The ROC curves of datasets trained on autoencoder models with a latent space of 4	26
4.6	The ROC curves of Phoneme and Pollen datasets trained on autoencoder models with latent spaces of 320 and 16 respectively	26
4.7	Correlation between latent space and epoch size	27
4.8	The influence of hyperparameters on an autoencoder model	28
4.9	The ROC curves of various datasets trained with autoencoder models	29
4.10	Comparison the average of AUC scores of autoencoder model, independent autoencoder ensemble and sequential autoencoder ensemble using different type of base learners	33
4.11	ROC AUC performance of each dataset relates to the number of base learners in a sequential autoencoder ensemble	33
4.12	Comparison the average of AUC scores of autoencoder model, independent autoencoder ensemble, sequential autoencoder ensemble and full sequential autoencoder ensemble using different type of base learners	35
4.13	ROC AUC performance of each dataset relates to the number of base learners in sequential autoencoder ensemble and full sequential autoencoder ensemble	35

4.14	Comparison the average of AUC scores of autoencoder model, independent autoencoder ensemble, sequential autoencoder ensemble, full sequential autoencoder ensemble, and synchronizing sequential autoencoder ensemble using different type of base learners	37
4.15	ROC AUC performance of each dataset relates to the number of base learners in sequential autoencoder ensemble, full sequential autoencoder ensemble, and independent autoencoder ensemble	41

List of Tables

4.1	List of datasets used in experiments and their size	21
4.2	Hyperparameters used for autoencoder models in experiments	23
4.3	Latent size used to train autoencoder models for each dataset	23
4.4	Hyperparameter values initialized for each type of base learners	29
4.5	Latent size used to train each type of base learners for each dataset	30
4.6	Comparison AUC scores of autoencoder models, independent autoencoder ensembles, and RandNet. The best AUC is highlighted in boldface	31
4.7	Comparison AUC scores of autoencoder model, independent autoencoder ensemble and sequential autoencoder ensemble. The best AUC is highlighted in boldface.	32
4.8	Comparison AUC scores of autoencoder model, independent autoencoder ensemble, sequential autoencoder ensemble, and full sequential autoencoder ensemble. The best AUC is highlighted in boldface.	34
4.9	Comparison AUC scores of autoencoder model, independent autoencoder ensemble, sequential autoencoder ensemble,full sequential autoencoder ensemble, synchronizing sequential autoencoder ensemble, and RandNet. The best AUC is highlighted in boldface.	36
4.10	The p-values obtained by implementing the Wilcoxon tests of sequential and full sequential autoencoder ensembles against independent autoencoder ensemble, the Wilcoxon test of sequential autoencoder ensemble against full sequential and the Wilcoxon tests of autoencoders against different ensembles	39

List of Algorithms

3.1	Sequential Autoencoder Ensemble Algorithm	17
3.2	Full Sequential Autoencoder Ensemble Algorithm	19
3.3	Synchronizing Sequential Autoencoder Ensemble Algorithm	20

Bibliography

- [1] *Tensorflow implementation of "Outlier Detection with Autoencoder Ensembles" by J. Chen, S. Sathe, C. Aggarwal and D. Turaga.* <https://github.com/psorus/RandNet>. Accessed: 2023-11-09.
- [2] AGGARWAL, CHARU C: *Outlier ensembles: position paper.* ACM SIGKDD Explorations Newsletter, 14(2):49–58, 2013.
- [3] AGGARWAL, CHARU C and CHARU C AGGARWAL: *An introduction to outlier analysis.* Springer, 2017.
- [4] AN, JINWON and SUNGZOOK CHO: *Variational autoencoder based anomaly detection using reconstruction probability.* Special lecture on IE, 2(1):1–18, 2015.
- [5] BANDYOPADHYAY, HMRISHAV: *Autoencoders in Deep Learning: Tutorial and Use Cases [2023].* <https://www.v7labs.com/blog/autoencoders-guide>. Accessed: 2023-07-30.
- [6] BANK, DOR, NOAM KOENIGSTEIN and RAJA GIRYES: *Autoencoders.* arXiv preprint arXiv:2003.05991, 2020.
- [7] BENGIO, YOSHUA, AARON COURVILLE and PASCAL VINCENT: *Representation learning: A review and new perspectives.* IEEE transactions on pattern analysis and machine intelligence, 35(8):1798–1828, 2013.
- [8] BÖING, BENEDIKT, SIMON KLÜTTERMANN and EMMANUEL MÜLLER: *Post-Robustifying Deep Anomaly Detection Ensembles by Model Selection.* In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 861–866. IEEE, 2022.
- [9] BREIMAN, LEO: *Bagging predictors.* Machine learning, 24:123–140, 1996.
- [10] BROWNLIE, JASON: *What is the Difference Between a Batch and an Epoch in a Neural Network.* Machine Learning Mastery, 20, 2018.
- [11] CHANDOLA, VARUN, ARINDAM BANERJEE and VIPIN KUMAR: *Anomaly detection: A survey.* ACM computing surveys (CSUR), 41(3):1–58, 2009.

- [12] CHARU, C AGGARWAL: *OUTLIER ANALYSIS*. springer, 2019.
- [13] CHAURASIA, SIDDHARTH, SAGAR GOYAL and MANISH RAJPUT: *Outlier detection using autoencoder ensembles: A robust unsupervised approach*. In *2020 International Conference on Contemporary Computing and Applications (IC3A)*, pages 76–80. IEEE, 2020.
- [14] CHEN, JINGHUI, SAKET SATHE, CHARU AGGARWAL and DEEPAK TURAGA: *Outlier detection with autoencoder ensembles*. In *Proceedings of the 2017 SIAM international conference on data mining*, pages 90–98. SIAM, 2017.
- [15] CHIANG, ALVIN, ESTHER DAVID, YUH-JYE LEE, GUY LESHEM and YI-REN YEH: *A study on anomaly detection ensembles*. Journal of Applied Logic, 21:1–13, 2017.
- [16] DAWID, ALEXANDER PHILIP and ALLAN M SKENE: *Maximum likelihood estimation of observer error-rates using the EM algorithm*. Journal of the Royal Statistical Society: Series C (Applied Statistics), 28(1):20–28, 1979.
- [17] DIETTERICH, THOMAS G: *Ensemble methods in machine learning*. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.
- [18] DONMEZ, PINAR, GUY LEBANON and KRISHNAKUMAR BALASUBRAMANIAN: *Unsupervised Supervised Learning I: Estimating Classification and Regression Errors without Labels*. Journal of Machine Learning Research, 11(4), 2010.
- [19] FREUND, YOAV and ROBERT E SCHAPIRE: *A desicion-theoretic generalization of on-line learning and an application to boosting*. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.
- [20] GOODFELLOW, IAN, YOSHUA BENGIO and AARON COURVILLE: *Deep learning*. MIT press, 2016.
- [21] HAN, JIAWEI, JIAN PEI and HANGHANG TONG: *Data mining: concepts and techniques*. Morgan kaufmann, 2022.
- [22] HAN, SONGQIAO, XIYANG HU, HAILIANG HUANG, MINQI JIANG and YUE ZHAO: *Adbench: Anomaly detection benchmark*. Advances in Neural Information Processing Systems, 35:32142–32159, 2022.
- [23] HANLEY, JAMES A and BARBARA J MCNEIL: *The meaning and use of the area under a receiver operating characteristic (ROC) curve*. Radiology, 143(1):29–36, 1982.
- [24] HANSEN, LARS KAI and PETER SALAMON: *Neural network ensembles*. IEEE transactions on pattern analysis and machine intelligence, 12(10):993–1001, 1990.

- [25] HASTIE, TREVOR, ROBERT TIBSHIRANI, JEROME H FRIEDMAN and JEROME H FRIEDMAN: *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.
- [26] HINTON, GEOFFREY E and RUSLAN R SALAKHUTDINOV: *Reducing the dimensionality of data with neural networks*. science, 313(5786):504–507, 2006.
- [27] HOLM, STURE: *A simple sequentially rejective multiple test procedure*. Scandinavian journal of statistics, pages 65–70, 1979.
- [28] JAFFE, ARIEL, ETHAN FETAYA, BOAZ NADLER, TINGTING JIANG and YUVAL KLUGER: *Unsupervised ensemble learning with dependent classifiers*. In *Artificial Intelligence and Statistics*, pages 351–360. PMLR, 2016.
- [29] KANDEL, IBRAHEM and MAURO CASTELLI: *The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset*. ICT express, 6(4):312–315, 2020.
- [30] KESKAR, NITISH SHIRISH, DHEEVATSA MUDIGERE, JORGE NOCEDAL, MIKHAIL SMELYANSKIY and PING TAK PETER TANG: *On large-batch training for deep learning: Generalization gap and sharp minima*. arXiv preprint arXiv:1609.04836, 2016.
- [31] KINGMA, DIEDERIK P and JIMMY BA: *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014.
- [32] PLATANIOS, EMMANOUIL ANTONIOS, AVRIM BLUM and TOM MITCHELL: *Estimating accuracy from unlabeled data*. 2014.
- [33] RAYKAR, VIKAS C, SHIPENG YU, LINDA H ZHAO, GERARDO HERMOSILLO VALADEZ, CHARLES FLORIN, LUCA BOGONI and LINDA MOY: *Learning from crowds*. Journal of machine learning research, 11(4), 2010.
- [34] RUFF, LUKAS, JACOB R KAUFFMANN, ROBERT A VANDERMEULEN, GRÉGOIRE MONTAVON, WOJCIECH SAMEK, MARIUS KLOFT, THOMAS G DIETTERICH and KLAUS-ROBERT MÜLLER: *A unifying review of deep and shallow anomaly detection*. Proceedings of the IEEE, 109(5):756–795, 2021.
- [35] RUMELHART, DAVID E, GEOFFREY E HINTON, RONALD J WILLIAMS et al.: *Learning internal representations by error propagation*, 1985.
- [36] SEWAK, MOHIT, SANJAY K SAHAY and HEMANT RATHORE: *An overview of deep learning architecture of deep neural networks and autoencoders*. Journal of Computational and Theoretical Nanoscience, 17(1):182–188, 2020.

- [37] SHAHAM, URI, XIUYUAN CHENG, OMER DROR, ARIEL JAFFE, BOAZ NADLER, JOSEPH CHANG and YUVAL KLUGER: *A deep learning approach to unsupervised ensemble learning*. In *International conference on machine learning*, pages 30–39. PMLR, 2016.
- [38] SHARMA, SAGAR, SIMONE SHARMA and ANIDHYA ATHAIYA: *Activation functions in neural networks*. *Towards Data Sci*, 6(12):310–316, 2017.
- [39] SRIVASTAVA, NITISH, GEOFFREY HINTON, ALEX KRIZHEVSKY, ILYA SUTSKEVER and RUSLAN SALAKHUTDINOV: *Dropout: a simple way to prevent neural networks from overfitting*. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [40] WELINDER, PETER, STEVE BRANSON, PIETRO PERONA and SERGE BELONGIE: *The multidimensional wisdom of crowds*. *Advances in neural information processing systems*, 23, 2010.
- [41] WIKIPEDIA: *Receiver operating characteristic*. https://en.wikipedia.org/wiki/Receiver_operating_characteristic. Accessed: 2023-08-16.
- [42] WILCOXON, FRANK: *Individual comparisons by ranking methods*. In *Breakthroughs in Statistics: Methodology and Distribution*, pages 196–202. Springer, 1992.
- [43] WOLPERT, DAVID H: *Stacked generalization*. *Neural networks*, 5(2):241–259, 1992.
- [44] ZHAO, HANG, ORAZIO GALLO, IURI FROSIO and JAN KAUTZ: *Loss functions for image restoration with neural networks*. *IEEE Transactions on computational imaging*, 3(1):47–57, 2016.
- [45] ZIMEK, ARTHUR, RICARDO JGB CAMPELLO and JÖRG SANDER: *Ensembles for unsupervised outlier detection: challenges and research questions a position paper*. *Acm Sigkdd Explorations Newsletter*, 15(1):11–22, 2014.
- [46] ZIMEK, ARTHUR, ERICH SCHUBERT and HANS-PETER KRIEGEL: *A survey on unsupervised outlier detection in high-dimensional numerical data*. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(5):363–387, 2012.

Eidesstattliche Versicherung

(Affidavit)

Nguyen, Phuong Huong

216601

Name, Vorname
(surname, first name)

Matrikelnummer
(student ID number)

☒ Bachelorarbeit
(Bachelor's thesis)

☐ Masterarbeit
(Master's thesis)

Titel
(Title)


Evaluating Sequential Autoencoder Ensemble for Anomaly Detection

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Dortmund, 16.10.2023

Ort, Datum
(place, date)


Unterschrift
(signature)

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:

Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).


The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Dortmund, 16.10.2023

Ort, Datum
(place, date)


Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**