UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Ingegneria "Enzo Ferrari"

Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

Unsupervised anomaly detection on time series data

Relatore
prof. Simone Calderara

Candidato:
Erminia de Vitis

Tutor
Axyon AI
dott. Jacopo Credi

Aprile 2023

A Silvia, Daniele, Camilla ed Alberto

Anomaly detection, anche conosciuta come outlier detection, è un processo che trova molteplici applicazioni, come ad esempio l'antiriciclaggio di denaro, la rilevazione di malattie rare, l'analisi dei social media e la segnalazione di intrusioni. In particolare, gli algoritmi di anomaly detection hanno lo scopo di individuare istanze che si discostano in modo significativo dalla maggior parte dei dati. L'argomento centrale di questa tesi è anomaly detection su serie temporali. Nel corso degli ultimi mesi, ho svolto un tirocinio curriculare presso Axyon AI, dove ho condotto ricerche e sperimentazioni che costituiscono la base di questo elaborato. Axyon AI collabora con gestori patrimoniali e fondi per fornire strategie di investimento basate sull'intelligenza artificiale, applicando il deep learning su serie temporali finanziarie. Tuttavia, il mercato finanziario è caratterizzato da cambiamenti continui nel tempo, il che può far sì che le prestazioni dei modelli supervisionati possano degradarsi e diventare meno affidabili. L'obiettivo principale di questo progetto è stato quello di sviluppare un detector di anomalie non supervisionato capace di stimare la confidenza della previsione. A tal fine, sono stati allenati in parallelo un classificatore e un detector utilizzando sia dati contaminati che dati puliti. L'anomaly detector è stato sviluppato con due modelli differenti, il Gaussian Mixture Model e successivamente un Variational Autoencoder. A seguire, è stato ricercato un legame tra lo score di uscita del classificatore e quello del detector. Una volta stabilita una significativa correlazione tra questi due score, è possibile utilizzarli tramite la regola di Bayes per ottenere uno score di confidenza della previsione. Questa tesi è suddivisa in cinque capitoli. Il primo capitolo offre un'introduzione al problema, al contesto in cui è stato affrontato e alle motivazioni che hanno portato alla realizzazione degli esperimenti. Il secondo capitolo presenta a livello teorico gli argomenti trattati e affrontati nella fase sperimentale. Nel terzo e quarto capitolo, si approfondiscono i metodi utilizzati e i risultati ottenuti. Infine, nel quinto capitolo, vengono presentati i miglioramenti che possono essere fatti per ottenere risultati più promettenti, basati sulle osservazioni ottenute dai risultati ottenuti durante la ricerca.

Acknowledgements

Un ringraziamento al prof.Calderara che con grande impegno e passione per il suo lavoro, mi ha guidato dandomi sostegno e incoraggiamento per tutto il percorso universitario, dalla triennale fino alla magistrale.

Un ringraziamento a Jacopo Credi, Simone Mistrali, Roberto Landi, Riccardo Folloni e tutto il team Axyon, per avermi dato un enorme supporto durante i mesi di tirocinio.

Un ringraziamento anche a Giovanni Dipierro del team G-Nous tech per la sua dedizione e la sua pazienza nel dedicare il suo tempo ad insegnarmi ogni giorno nuove cose.

Summary

Anomaly detection, which is also known as outlier detection, is a task with numerous applications, including anti-money laundering, rare disease detection, social media analysis , and intrusion detection. Anomaly detection algorithms aim to identify data instances that deviate significantly from the majority of data objects. The central topic of this thesis is anomaly detection performed on time series. For the past few months, I have been an intern at Axyon AI. Axyon AI partners with asset managers and hedge funds to deliver consistently high-performing AI-powered investment strategies by leveraging its proprietary, fully automated deep learning technology for financial time series. Supervised learning models are trained to find patterns and regularities in the data in order to predict a future value. Supervised models are able to make predictions about data that have never been seen, because they extract knowledge from patterns and regularities in the data on which they are trained. Financial market is subject to frequent mutations over time, for this reason the performance of a supervised model may not be reliable in prediction. The main goal of this work is to develop an unsupervised anomaly detector capable of estimating prediction confidence. A classifier and an anomaly detector were trained simultaneously, using clean data and contaminated data. To be more specific, After this training phase, the focus shifted to looking for a correlation between the score coming out of the classifier and the score coming out of the detector. The anomaly detector was developed with two different models, first Gaussian Mixture model, later Variational Autoencoder. Given these two scores, through Bayes' rule, a confidence score of the prediction would be calculated. Given the confidence score, one could discard the points with a low confidence score to denoise the classifier or simply notify these points as edge cases in the classification problem. The thesis is organized in five chapters. The first one is an introduction to the problem, the context in which the problem is addressed, and the reasons behind the development of the experiments. The second one refers to the literature review of the topics covered throughout the work. In the third and fourth I go into the merits of, respectively, the methods followed and the results obtained. Given the observations made on the results, the last chapter presents the improvements that could be made to obtain more promising results.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivations

Time series are used in a large number of fields such as industrial control systems, finance, and healthcare. Time series and their analysis are becoming increasingly important due to the massive production of these data. Time series analysis is a specific way of analysing a sequence of data points collected over an interval of time. Time is a crucial variable because it shows how the data adjusts over the course of the data points and it provides an additional source of information and a set order of dependencies between the data. Time series analysis typically requires a large number of data points to ensure consistency and reliability. An extensive data set ensures that any trends or patterns discovered are not outliers and can account for seasonal variance.
Detecting unexpected behaviours or patterns that do not conform to the expected behaviour is an active research discipline called anomaly detection in time series. Anomaly detection is an important field. It consists in detecting rare events or, more generally, observations that are aberrant and different from the majority of data. These rare events can be of various types and they are present in multiple and different domains (fraudulent financial transactions, medical problems or network intrusions). Detecting these rare events is a major issue for many fields.
In the first place this thesis focuses on the critical task of anomaly detection. Specifically, it focuses on a subset of time series anomaly detection methods, which is unsupervised detection. Unlike supervised detection, unsupervised detection methods do not require a label associated with the data samples. The ultimate goal is to obtain, through the implementation of anomaly detection, a calibrated classifier that can make a prediction with the right level of confidence.
Calibration is comparison of the actual output and the expected output given by a system. In a calibrated model the distribution and the behaviour of the probability predicted is similar to the distribution and behaviour of probability observed in training data. Confidence calibration is defined as the ability of some model to provide an accurate probability of correctness for any of its predictions. Such calibrated confidence scores are important in various “high-stakes” applications where incorrect predictions are extremely problematic (e.g., self-driving cars, medical diagnosis, etc.), as calibrated probability scores associated with each prediction allow low-quality predictions to be

identified and discarded.

1.2 Contributions

The work explained in this thesis was developed during the curricular internship at Axyon AI. Axyon AI is a fintech company with a mission to bring AI-powered predictive value to the investment management industry. Axyon partners with asset managers and hedge funds to deliver consistently high-performing AI-powered investment strategies by leveraging its proprietary, fully automated AI/deep learning technology for financial time series. The goal of the work in Axyon AI was to deepen and research the topic of anomaly detection, experiment on the data and models they already used, in order to be able to improve the confidence of the predictions made by their model.

1.3 Structure of the Thesis

This thesis is structured as follows:

- The current chapter, i.e., the first chapter, is an introductory chapter in which I introduce the background and motivation behind this project.
- The second chapter reviews the literature on each of the topics and methodologies applied and developed in the experimental phase of the project.
- The third chapter presents the method and mode used in the pilot phase of the project.
- The fourth chapter presents the results obtained in the various steps.
- The last chapter presents final observations consequent to the results obtained and possible future approaches and improvements to the central topic of this project.

Chapter 2

Literature Review

2.1 Anomaly detection

An anomaly can be defined as an observation that is unexpected with respect to a set of other pre established observations considered as normal. More formally, in a set D containing n observations noted x_i , then $x_p \in A$ will be considered as abnormal if it differs, by its characteristics, from the other observations. The definition of the term anomaly is specific to the use case. The most common one in the field of detection is an observation which is different from the others by its singularity: it could result from a set of rules which are different from the other observations. Anomalies in time series, also called outliers, are points or sequences of points that do not correspond to normal behaviour. The concept of normal behaviour is difficult to formalise. Therefore, another possible definition for anomalies could be a pattern in data that is not expected in comparison to what has been seen before. In fact, an implicit assumption is that anomalies are rare events. Anomalies should not be confused with the noise present in the time series. Anomaly detection refers to the task of identifying an unseen observation x_t , $t > \tau$, based on the fact that it differs significantly from τ, thus assuming that τ contains only normal points. The amount by which the unseen sample x_t and the normal set τ differ is measured by an anomaly score, which is then compared to a threshold to obtain an anomaly label. Three broad categories of anomaly detection techniques exist.

- Supervised anomaly detection techniques require a data set that has been labelled as "normal" and "abnormal" and involves training a classifier. However, this approach is rarely used in anomaly detection due to the general unavailability of labelled data and the inherent unbalanced nature of the classes.
- Semi-supervised anomaly detection techniques assume that some portion of the data is labelled. This may be any combination of the normal or anomalous data, but more often than not the techniques construct a model representing normal behaviour from a given normal training data set, and then test the likelihood of a test instance to be generated by the model.
- Unsupervised anomaly detection techniques assume the data is unlabelled and are by far the most commonly used due to their wider and relevant application.

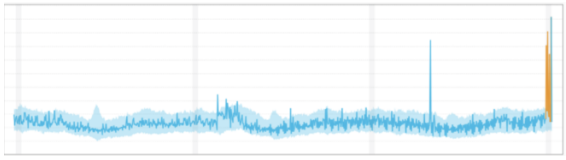
In this thesis, only unsupervised techniques of anomaly detection were addressed. The goal of unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. In supervised anomaly detection, if we want a model to be able to detect anomalies, it must characterise the system very precisely both in normal behaviour and in the presence of anomalies. However, normal behaviours can be multiple, as well as behaviours in the presence of anomalies. Unsupervised learning is perfectly adapted to the problem of anomaly detection since it is not necessary to label large data sets. Moreover, a part of the anomalies come from new behaviours of the system. By definition, these behaviours could not be correctly classified with supervised anomaly detection methods.

2.2 Types of time series anomalies

Understanding the types of outliers that an anomaly detection system can identify is essential to getting the most value from generated insights. Without knowing what you're up against, you risk making the wrong decisions once your anomaly detection system alerts you to an issue or opportunity. Generally speaking, anomalies in data fall into three main outlier categories — global outliers, contextual outliers, and collective outliers.

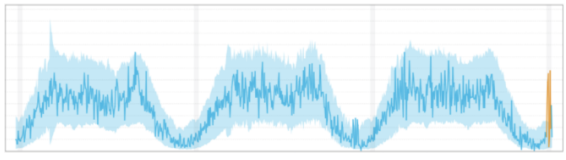
- Global outliers: This is the simplest type of anomaly. It corresponds to a point that differs from the rest of the data.

Figure 2.1: Global outliers example



- Contextual outliers: Also called conditional outliers, these anomalies have values that significantly deviate from the other data points that exist in the same context. An anomaly in the context of one dataset may not be an anomaly in another. These outliers are common in time series data because those datasets are records of specific quantities in a given period. The value exists within global expectations but may appear anomalous within certain seasonal data patterns.

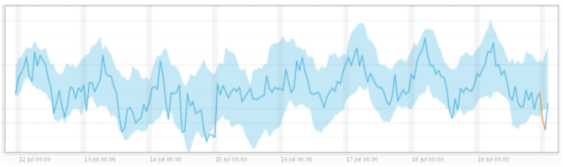
Figure 2.2: Contextual outliers example



- Collective outliers: When a subset of data points within a set is anomalous to the entire dataset, those values are called collective outliers. In this category, individual

values are not anomalous globally or contextually. You start to see these types of outliers when examining distinct time series together. Individual behaviour may not deviate from the normal range in a specific time series dataset. But when combined with another time series dataset, more significant anomalies become clear.

Figure 2.3: Collective outliers example



2.3 Latent anomaly detection

In this work, a different approach to anomaly detection has been proposed, the first step is looking at the latent variable space to make toward anomaly detection. Most conventional approaches to anomaly detection are concerned with tracking data which are largely deviated from the ordinary pattern. In this case, the issue is track changes happening in the latent variable space consisting of the meta information existing behind observed data.
The key idea is estimates the $p(x)$ of the data using a class-agnostic unsupervised method, in particular in this work was used Gaussian Mixture Model and Variational Autoencoder. What is expected is that in case of anomalies the $p(x)$ estimates are higher than the $p(x)$ estimates for normal data.

2.3.1 Latent variable models

Latent variable modeling refers to a varied group of statistical procedures that use one or more unobserved (latent) variables to explain and explore relationships between a larger set of observed variables.
Latent variable models take an indirect approach to describing a probability distribution $P_r(x)$ over a multi-dimensional variable x. Instead of directly writing the expression for $P_r(x)$ they model a joint distribution $P_r(x, h)$ of the data x and an unobserved latent (or hidden) variable h. They then describe the probability of $P_r(x)$ as a marginalization of this joint probability so that

$$P_r(x) = \int P_r(x, h)dh \tag{2.1}$$

Typically we describe the join probability $P_r(x, h)$ as the product of the *likelihood* $P_r(x|h)$ and the *prior* $P_r(h)$, so that the model becomes

$$P_r(x) = \int P_r(x|h)P_r(h)dh \tag{2.2}$$

It is reasonable to question why we should take this indirect approach to describing $P_r(x)$. The answer is that relatively simple expressions for $P_r(x|h)$ and $P_r(h)$ can describe a very complex distribution for $P_r(x)$.

2.3.2 Gaussian Mixture Models

A well known latent variable model is the mixture of Gaussians, a weighted sum of C Gaussian components.

$$p(x|\theta) = \sum_{c=1}^{C} \pi_c N(x|\pi_c, \Sigma_c) \tag{2.3}$$

π_c,μ_c and Σ_c are are the weight, mean vector and covariance matrix of mixture component c, respectively. The weights are non-negative and sum up to 1 i.e. $\sum_{c=1}^{C} \pi_c = 1$. Parameters vector $\theta = \pi_1, \mu_1, \Sigma_1, ..., \pi_c, \mu_c, \Sigma_c$ denotes the set of all model parameters. If we introduce a discrete latent variable t that determines the assignment of observations to mixture components we can define a joint distribution over observed and latent variables $p(x, t|\theta)$ in terms of a conditional distribution $p(x|t, \theta)$ and a prior distribution $p(t|\theta)$

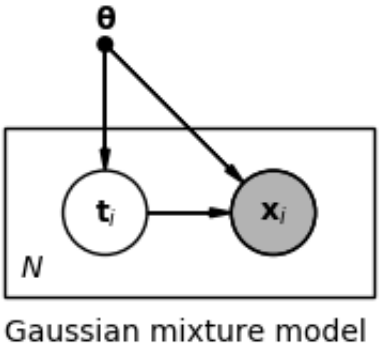
$$p(x, t|\theta) = p(x|t, \theta)p(t|\theta) \tag{2.4}$$

dove $p(x|t_c = 1, \theta) = N(x|\mu_c, \Sigma_c)$ and $p(t_c = 1|\theta)$. The marginal distribution $p(x|\theta)$ is obtained by summing over all possible states of t.

$$p(x|\theta) = \sum_{c=1}^{C} p(t_c = 1|\theta)p(x|t_c = 1, \theta) = \sum_{c=1}^{C} N(x|\mu_c, \Sigma_c) \tag{2.5}$$

For each observation x_i we have one latent variable t_i, as shown in the following plate notation of the model.

Figure 2.4: Gaussian mixture model



We denote the set of all observations by X and the set of all latent variables by T. If we could observe T directly, we could easily maximize the complete-data likelihood $p(X, T|\theta)$ because we would then know the assignment of data points to components, and fitting a single Gaussian per component can be done analytically. But since we can only observe X, we have to maximize the marginal likelihood or incomplete-data likelihood $p(X|\theta)$. By

using the logarithm of the likelihood we have

$$\theta_{MLE} = \underset{\theta}{\operatorname{argmax}} log p(X|\theta) = \underset{\theta}{\operatorname{argmax}} log \sum_{T} p(X, T|\theta) \tag{2.6}$$

which involves a summation over latent variables inside the logarithm. This prevents a simple analytical solution to the optimization problem.

Expectation maximization algorithm

Although we cannot observe latent variables directly, we can obtain their posterior distribution. We start with a preliminary parameter value θ^{old}.

$$p(T|X, \theta^{old}) = \frac{p(X|T, \theta^{old})p(T|\theta^{old})}{\sum_T p(X|T, \theta^{old})p(T|\theta^{old})} \tag{2.7}$$

This allows us to define an expectation of the complete-data likelihood w.r.t. to the posterior distribution.

$$Q(\theta, \theta^{old}) = \sum_{T} p(T|X, \theta^{old}) \log p(X, T|\boldsymbol{\theta}) = \mathbb{E}_{p(T|X,\theta^{old})} \log p(X, T\theta) \tag{2.8}$$

This expectation is then maximized w.r.t. to θ resulting in an updated parameter vector θ^{new}.

$$\theta^{new} = \underset{\theta}{\operatorname{argmax}} \mathcal{Q}(\theta, \theta^{old}) \tag{2.9}$$

In eq.(2.8) the summation is outside the logarithm which enables an analytical solution for θ^{new} in the case of GMMs. We then let $\theta^{old} \leftarrow \theta^{new}$ and repeat these steps until convergence. This is the essence of the expectation maximization (EM) algorithm. It has an expectation- or E-step where the posterior over latent variables is obtained and a maximization- or M-step where the expectation of the complete-data likelihood w.r.t. the posterior distribution is maximized. It can be shown that the EM algorithm always converges to a local maximum of $p(X|\theta)$. By introducing a latent variable t_i for each observation x_i we can define the log marginal likelihood as

$$\log p(X|\theta) = \sum_{i=1}^{N} \log p(x_i|\theta) = \sum_{i=1}^{N} \log \sum_{c=1}^{C} p(x_i, t_{ic} = 1|\theta) \tag{2.10}$$

Next we introduce a distribution $q(t_i)$ over latent variable t_i.

$$\log p(X|\theta) = \sum_{i=1}^{N} \log \sum_{c=1}^{C} q(t_{ic} = 1) \frac{p(x_i, t_{ic} = 1|\theta)}{q(t_{ic} = 1)} = \sum_{i=1}^{N} \log \mathbb{E}_{q(t_i)} \frac{p(x_i, t_i|\theta)}{q(t_i)} \tag{2.11}$$

We now have a concave function log of an expectation which allows us to apply Jensen's inequality to define a lower bound $\mathcal{L}$ on $\log p(X|\theta)$.

$$\begin{aligned} \log p(X|\theta) &= \sum_{i=1}^{N} \log \mathbb{E}_{q(t_i)} \frac{p(x_i, t_i|\theta)}{q(t_i)} \geq \sum_{i=1}^{N} \mathbb{E}_{q(t_i)} \log \frac{p(x_i, t_i|\theta)}{q(t_i)} \\ &= \mathbb{E}_{q(T)} \log \frac{p(x, T|\theta)}{q(T)} \\ &= \mathcal{L}(\theta, q) \end{aligned}$$

This lower bound is a function of θ and q. When we subtract the lower bound from the log marginal likelihood we should end up with something that is non-negative.

$$\begin{aligned}\log p(X|\theta) - \mathcal{L}(\theta, q) &= \log p(X|\theta) - \mathbb{E}_{q(T)} \log \frac{p(X,T|\theta)}{q(T)} \\ &= \mathbb{E}_{q(T)} \log \frac{p(X|\theta)q(T)}{p(X,T|\theta)} \\ &= \mathbb{E}_{q(T)} \log \frac{q(T)}{p(T|X,\theta)} \\ &= \mathrm{KL}(q(T) \mid\mid p(T|X,\theta))\end{aligned}$$

We end up with the Kullback-Leibler (KL) divergence between $q(T)$ and the true posterior over latent variables. It can be shown that the KL divergence is always non-negative. We finally can write the following expression for the lower bound.

$$\mathcal{L}(\theta, q) = \log p(X|\theta) - \mathrm{KL}(q(T) \mid\mid p(T|X,\theta)) \tag{2.12}$$

In the E-step of the EM algorithm $\mathcal{L}(\theta, q)$ is maximized w.r.t. q and θ is held fixed.

$$q^{new} = \underset{q}{\mathrm{argmax}} \mathcal{L}(\theta^{old}, q) = \underset{q}{\mathrm{argmin}} \mathrm{KL}(q(T) \mid\mid p(T|X,\theta^{old})) \tag{2.13}$$

$\mathcal{L}(\theta, q)$ is maximized when $\mathrm{KL}(q(T) \mid\mid p(T|X,\theta))$ is minimized as $\log p(X|\theta)$ doesn't depend on q. If we can obtain the true posterior, like in the GMM case, we can set $q(T)$ to $p(T|X,\theta)$ and the KL divergence becomes 0. In the M-step $\mathcal{L}(\theta, q)$ is maximized w.r.t. θ and q is held fixed. Using Eq. (2.12) we get

$$\begin{aligned}\theta^{new} &= \underset{\theta}{\mathrm{argmax}} \mathcal{L}(\theta, q^{new}) \\ &= \underset{\theta}{\mathrm{argmax}} \mathbb{E}_{q^{new}(T)} \log \frac{p(X,T|\theta)}{q^{new}(T)} \\ &= \underset{\theta}{\mathrm{argmax}} \mathbb{E}_{q^{new}(T)} \log p((X,T|\theta) - \mathbb{E}_{q^{new}(T)} \log q^{new}(T) \\ &= \underset{\boldsymbol{\theta}}{\mathrm{argmax}} \mathbb{E}_{q^{new}(T)} \log p(X,T|\theta) + \mathrm{const.}\end{aligned}$$

If the true posterior is known Eq. (15) becomes Eq. (7) except for the constant term which can be ignored during optimization. Again, we let $\theta^{old} \leftarrow \theta^{new}$ and repeat these steps until convergence.

2.3.3 Variational Autoencoder

The goal of the variational autoencoder (VAE) is to learn a probability distribution $Pr(\mathbf{x})$ over a multi-dimensional variable $\mathbf{x}$. There are two main reasons for modelling distributions. First, we might want to draw samples (generate) from the distribution to create new plausible values of $\mathbf{x}$. Second, we might want to measure the likelihood that a new vector was created by this probability distribution. It is common to talk about the

variational autoencoder as if it is the model of $Pr(\mathbf{x})$. However, this is misleading; the variational autoencoder is a neural architecture that is designed to help learn the model for $Pr(\mathbf{x})$. The final model contains neither the 'variational' nor the 'autoencoder' parts and is better described as a non-linear latent variable model. Maximum likelihood learning of this model is not straightforward, but we can define a lower bound on the likelihood. To maximize the bound, derviatives need to be computed, but unfortunately, it's not possible to compute the derivative of the sampling component. To side-step this problem there is a trick called reparameterization trick.
The likelihood $Pr(\mathbf{x}|\mathbf{h}, \boldsymbol{\phi})$ allows to compute the distribution over the observed data given hidden variable $\mathbf{h}$. Moving in the other direction mean given an observed data example $\mathbf{x}$ which are the possible values of the hidden variable $\mathbf{h}$ were responsible for it. This information is encompassed in the posterior distribution, using Bayes's rule:

$$Pr(\mathbf{h}|\mathbf{x}) = \frac{Pr(\mathbf{x}|\mathbf{h})Pr(\mathbf{h})}{Pr(\mathbf{x})} \tag{2.14}$$

In practice, there is no closed form expression for the left hand side of this equation. The denominator $Pr(\mathbf{x})$ can not be evaluated and so the numerical value of the posterior for a given pair $\mathbf{h}$ and $\mathbf{x}$.
In order to solve the closed form expression, the posterior is approximated using a variational distribution. The ojective becomes approximate at best the posterior using a variational distribution $q(\mathbf{h})$.

$$\begin{aligned} \log[Pr(\mathbf{x}|\boldsymbol{\phi})] &= \log\left[\int Pr(\mathbf{x}, \mathbf{h}|\boldsymbol{\phi}) d\mathbf{h}\right] && (2.15) \\ &= \log\left[\int q(\mathbf{h}) \frac{Pr(\mathbf{x}, \mathbf{h}|\boldsymbol{\phi})}{q(\mathbf{h})} d\mathbf{h}\right], && (2.16) \end{aligned}$$

Given the Jensen's inequality for the logarithm:

$$log\left[\int q(\mathbf{h}) \frac{Pr(\mathbf{x}, \mathbf{h}|\boldsymbol{\phi})}{q(\mathbf{h})} d\mathbf{h}\right] \geq \int q(\mathbf{h}) \log\left[\frac{Pr(\mathbf{x}, \mathbf{h}|\boldsymbol{\phi})}{q(\mathbf{h})}\right] d\mathbf{h}, \tag{2.17}$$

where the term on the right hand side is known as the *evidence lower bound* or ELBO. In practice, the distribution $q(\mathbf{h})$ will have some parameters $\boldsymbol{\theta}$ as well and so the ELBO can be written as:

$$\text{ELBO}[\boldsymbol{\theta}, \boldsymbol{\phi}] = \int q(\mathbf{h}|\boldsymbol{\theta}) \log\left[\frac{Pr(\mathbf{x}, \mathbf{h}|\boldsymbol{\phi})}{q(\mathbf{h}|\boldsymbol{\theta})}\right] d\mathbf{h}. \tag{2.18}$$

To learn the non-linear latent variable model, this quantity is maximized as a function of both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$. The neural architecture that computes this quantity is the variational autoencoder.
The ELBO is described as being tight when for a fixed value of $\boldsymbol{\phi}$ some parameters $\boldsymbol{\theta}$ are chosen so that the ELBO and the likelihood function coincide. This happens when the

distribution $q(\boldsymbol{\theta})$ is equal to the posterior distribution $Pr(\mathbf{h}|\mathbf{x})$ over the hidden variables.

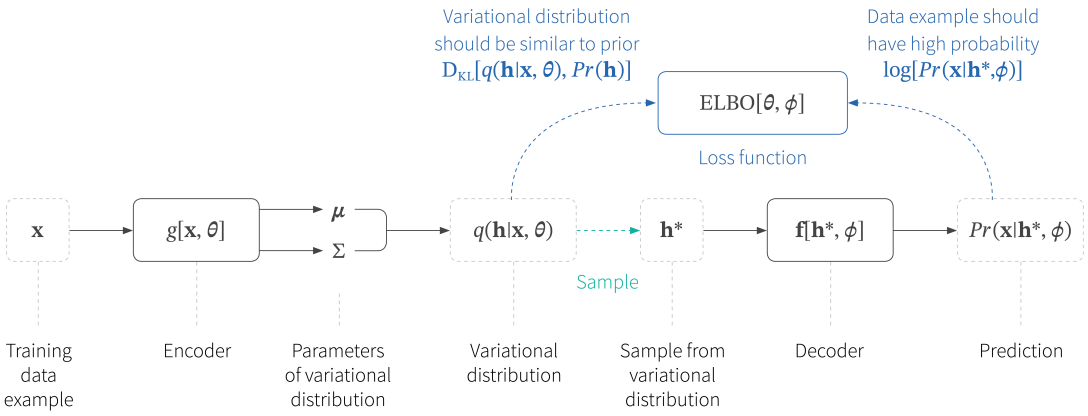
$$\begin{aligned}
\text{ELBO}[\boldsymbol{\theta}, \boldsymbol{\phi}] &= \int q(\mathbf{h}|\boldsymbol{\theta}) \log\left[\frac{Pr(\mathbf{x}, \mathbf{h}|\boldsymbol{\phi})}{q(\mathbf{h}|\boldsymbol{\theta})}\right] d\mathbf{h} \\
&= \int q(\mathbf{h}|\boldsymbol{\theta}) \log\left[\frac{Pr(\mathbf{h}|\mathbf{x}, \boldsymbol{\phi})Pr(\mathbf{x}|\boldsymbol{\phi})}{q(\mathbf{h}|\boldsymbol{\theta})}\right] d\mathbf{h} \\
&= \int q(\mathbf{h}|\boldsymbol{\theta}) \log\left[Pr(\mathbf{x}|\boldsymbol{\phi})\right] d\mathbf{h} + \int q(\mathbf{h}|\boldsymbol{\theta}) \ \log\left[\frac{Pr(\mathbf{h}|\mathbf{x}, \boldsymbol{\phi})}{q(\mathbf{h}|\boldsymbol{\theta})}\right] d\mathbf{h} \\
&= \log[Pr(\mathbf{x}|\boldsymbol{\phi})] + \int q(\mathbf{h}|\boldsymbol{\theta}) \ \log\left[\frac{Pr(\mathbf{h}|\mathbf{x}, \boldsymbol{\phi})}{q(\mathbf{h}|\boldsymbol{\theta})}\right] d\mathbf{h} \\
&= \log[Pr(\mathbf{x}|\boldsymbol{\phi})] - \mathrm{D}_{KL}\left[q(\mathbf{h}|\boldsymbol{\theta})||Pr(\mathbf{h}|\mathbf{x}, \boldsymbol{\phi})\right]. \qquad (2.19)
\end{aligned}$$

This equation shows that the ELBO is the original log likelihood minus the Kullback-Leibler divergence $\mathrm{D}_{KL}\left[q(\mathbf{h}|\boldsymbol{\theta})||Pr(\mathbf{h}|\mathbf{x}, \boldsymbol{\phi})\right]$ which will be zero when these distributions are the same. Hence the bound is tight when $q(\mathbf{h}|\boldsymbol{\theta}) = Pr(\mathbf{h}|\mathbf{x}, \boldsymbol{\phi})$. Since the KL divergence can only take non-negative values it is easy to see that the ELBO is a lower bound on $\log[Pr(\mathbf{x}|\boldsymbol{\phi})]$ from this formulation.
Given the fact that the posterior distribution $Pr(\mathbf{h}|\mathbf{x})$ over the hidden variables can not be expressed for non-linear latent model. The solution to the problem is to make a variational approximation: given a simple parametric form for $q(\mathbf{h}|\boldsymbol{\theta})$ and use this as an approximation to the true posterior. In this case a normal distribution with parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ is chosen. This distribution is not always going to be a great match to the posterior, but will be better for some values of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ than others. The model is optimized when is found the normal distribution that is "closest" to the true posterior $Pr(\mathbf{h}|\mathbf{x})$. This corresponds to minimizing the KL divergence.
Since the optimal choice for $q(\mathbf{h}|\boldsymbol{\theta})$ was the posterior $Pr(\mathbf{h}|\mathbf{x})$ and this depended on the data example $\mathbf{x}$, it makes sense that variational approximation should do the same and so:

$$q(\mathbf{h}|\boldsymbol{\theta}, \mathbf{x}) = \mathrm{Norm}_{\mathbf{h}}[g_{\mu}[\mathbf{x}|\boldsymbol{\theta}], g_{\sigma}[\mathbf{x}|\boldsymbol{\theta}]] \qquad (2.20)$$

where $g[\mathbf{x}, \boldsymbol{\theta}]$ is a neural network with parameters $\boldsymbol{\theta}$ that predicts the mean and variance of the normal variational approximation.

Figure 2.5: Variational autoencoder

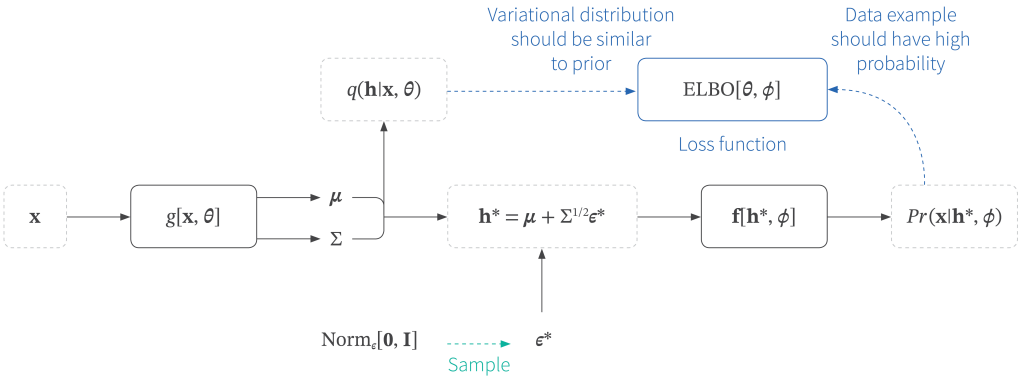


However, there's a problem. The network involves a sampling step and there is no way to differentiate through this. Consequently, it's impossible to make updates to the parameters $\boldsymbol{\theta}$ that occur earlier in the network than this. Fortunately, there is a simple solution; moving the stochastic part into a branch of the network which draws a sample from $\text{Norm}_{\epsilon}[\mathbf{0}, \mathbf{I}]$ and then use the relation

$$\mathbf{h}^* = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{1/2}\epsilon \tag{2.21}$$

to draw from the intended Gaussian. Now it is possible compute the derivatives as usual because there is no need for the backpropagation algorithm to pass down the stochastic branch. This is known as the reparameterization trick and is illustrated in figure.

Figure 2.6: Reparameterization trick



2.4 Time series analysis

This section gives techniques for time series analysis. It focuses on the problem of processing and analysing data to gain useful information. The methods presented are trend estimation and extraction of features.
Time series analysis is a specific way of analyzing a sequence of data points collected over an interval of time. In time series analysis, analysts record data points at consistent intervals over a set period of time rather than just recording the data points intermittently or randomly. However, this type of analysis is not merely the act of collecting data over time.
What sets time series data apart from other data is that the analysis can show how variables change over time. In other words, time is a crucial variable because it shows how the data adjusts over the course of the data points as well as the final results. It provides an additional source of information and a set order of dependencies between the data. Additionally, time series data can be used for forecasting—predicting future data based on historical data.
Time series analysis helps organizations understand the underlying causes of trends or systemic patterns over time. When organizations analyze data over consistent intervals, they can also use time series forecasting to predict the likelihood of future events. Time series forecasting is part of predictive analytics. It can show likely changes in the data,

like seasonality or cyclic behavior, which provides a better understanding of data variables and helps forecast better.
For example, Axyon AI srl Axyon AI is one of the leading players in deep learning solutions for time series forecasting in both traditional and decentralized finance.

2.4.1 Trend estimation

To analyse long term changes in a time series it can be useful to calculate the trend of the time series. The computed trend is itself a time series that explains underlying tendencies and can be viewed as a smoothed version of the original time series. There are multiple ways of computing the trend, this section presents the methods *moving average, moving median* and *exponentially weighted moving average*.

- **Moving Average** One of the most intuitive ways of computing the trend is to use moving average. Utilising this method the trend component, $\tilde{y}(t)$, at each point in time is the average of the n previous points. Formally, let y be a time series of process values and n be the number of previous points to use. The trend component or moving average at time t is given by

$$\tilde{y} = MA(t) = \frac{y_t + y_{t-1} + ... + y_{t-n}}{n} = \frac{1}{n}\sum_{i=t}^{t-n} y_y \tag{2.22}$$

- **Moving Median** An other intuitive way to calculate the trend is to use moving median. This is analogous to the moving average method, but the average is exchanged for the median. Formally, let y be a time series of process values and n be a fixed time frame. Then the trend component or moving median at time t is given by the median of the t previous points,

$$\tilde{y} = MM(t) = median(y_t + y_{t-1} + ... + y_{t-n}). \tag{2.23}$$

- **Exponentially weighted moving average**Another way of calculating a trend is to use an exponentially weighted moving average. This method is related to the moving average, but uses a smoothing coefficient θ instead of a number of previous points n. The exponentially moving average is the average of all historical points but the influence of the historical points decay exponentially with time. Let y_t be the value of the time series at time t and $\theta \in [0,1]$ be the smoothing constant. The exponentially weighted average is given by

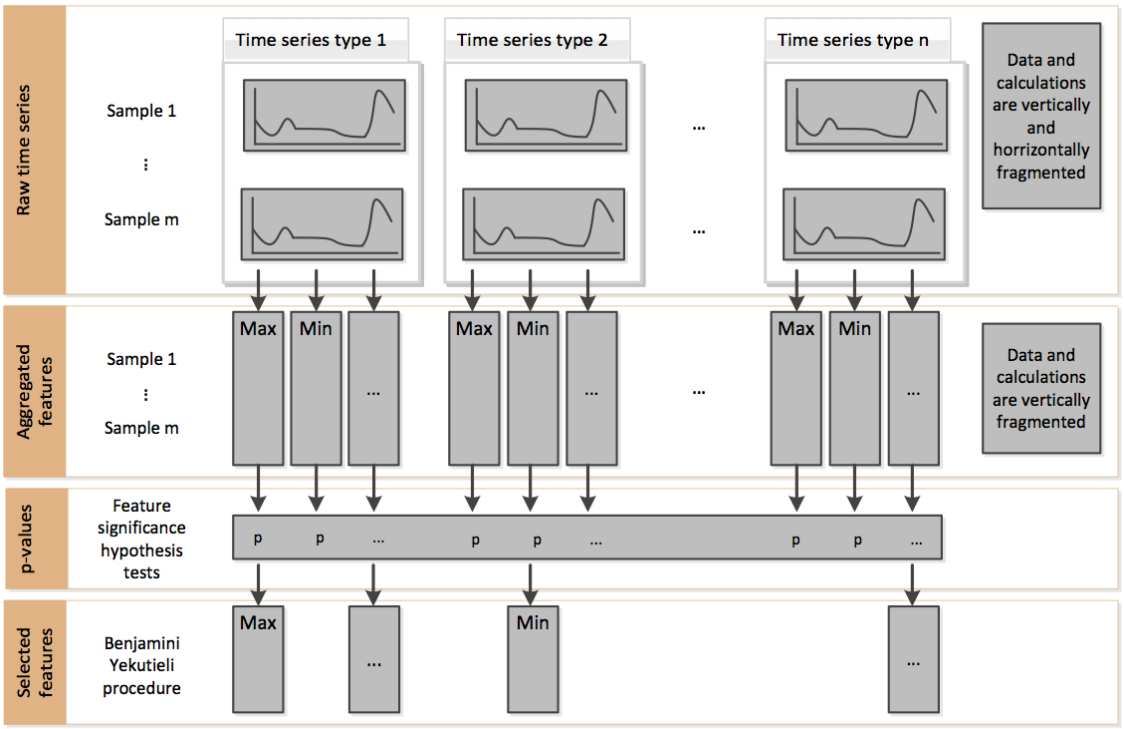
$$\tilde{y} = EWMA(t) = (1-\theta)(y_t + \theta y_{t-1} + \theta^2 y_{t-n} + ...). \tag{2.24}$$

2.4.2 Extraction of features

A time series consists, in its raw state, of a collection of time stamps and values associated with these. To detect changes in amplitude or shape of the time series it is necessary to extract information about the patterns of the time series. This section defines the important task of features engineering in time series and introduce `FRESH` (FeatuRe Extraction and Scalable Hypothesis testing) algorithm. Feature engineering

plays a crucial role in many of the data modelling tasks. This is simply a process that defines important features of the data using which a model can enhance its performance. In time series modelling, feature engineering works in a different way because it is sequential data and it gets formed using the changes in any values according to the time. In a popular study Maximilian Christ et al. (2016) proposed an algorithm that combines stablished feature extraction methods with a feature importance filter. `FRESH` is an efficient, scalable feature extraction algorithm, which filters the available features in an early stage of the machine learning pipeline with respect to their significance for the classification or regression task, while controlling the expected percentage of selected but irrelevant features. The algorithm characterizes time series with comprehensive and well-established feature mappings and considers additional features describing meta-information. In a second step, each feature vector is individually and independently evaluated with respect to its significance for predicting the target under investigation. The result of these tests is a vector of p-values, quantifying the significance of each feature for predicting the label/target. In order to characterize a time series with respect to its dynamics and reduce the data volume, a mapping $\theta k : \mathbb{R}^{n_t} \rightarrow \mathbb{R}$ is introduced, which captures a specific aspect k of the time series. One example for such mapping might be the maximum operator $\theta_{max}(s_{i,j}) = max(s_{i,j,1}, s_{i,j,2}, ..., s_{i,j,v}, ..., s_{i,j,n_t})$ which quantifies the maximal value ever recorded for time series $s_{i,j}$. This kind of lower dimensional representation is called a feature, which is a measurable characteristics of the considered time series. Other examples for feature mappings θk of time series might be their mean, the number of peaks with a certain steepness, their periodicity, a global trend, etc.

Figure 2.7: FRESH Algorithm



Feature filtering

Typically, time series are noisy and contain redundancies. Therefore, one should keep the balance between extracting meaningful but probably fragile features and robust but probably non-significant features. Some features such as the median will not be heavily influenced by outliers, others such as max will be intrinsically fragile. The choice of the right time series feature mappings is crucial to capture the right characteristics for the task at hand. A meaningless feature describes a characteristic of the time series that is not useful for the classification or regression task at hand. Given a binary target Y , stating that the relevance of feature X is measured as the difference between the class conditional distributions $f_{x|y=0}$ and $f_{x|y=1}$. In general, a feature X is relevant for predicting target Y if and only if (Radivojac et al. [2004])

$$\exists y_1, y_2 with f_Y(y_1) > 0, f_Y(y_2) > 0 : f_{x|y=y_1} \neq f_{x|y=y_2} \tag{2.25}$$

2.5 Confidence calibration

Confidence calibration is defined as the ability of some model to provide an accurate probability of correctness for any of its predictions. Such calibrated confidence scores are important in various "high-stakes" applications where incorrect predictions are extremely problematic (e.g., self-driving cars, medical diagnosis, etc.), as calibrated probability scores associated with each prediction allow low-quality predictions to be identified and discarded. Thus, even if neural network output cannot yet be fully explained, confidence calibration provides a practical avenue for avoiding major mistakes in practice by associating each prediction with an accurate uncertainty/confidence score.
The types of applications that depend most on properly-calibrated uncertainty.

- **Filtering poor predictions** iven a properly-calibrated model, predictions with high uncertainty (or low confidence) can be discarded to avoid unnecessary model errors. Such an ability to discard incorrect predictions based upon a confidence score associated with each prediction is especially impactful in the high-risk applications mentioned above. Although truly explaining or understanding neural network output may remain difficult in the near term, properly calibrated uncertainty scores provide a practical avenue for detecting and avoiding a neural network's mistakes.

- **Model-aware active learning** Proper uncertainty estimates enable data that is poorly-understood by the model to be easily identified. As a result, data associated with low-confidence predictions can be set aside and passed to a human annotator to be labeled and included in the model's training set. In this way, model uncertainty can be used to iteratively identify data that the model doesn't understand, enabling a form of model-aware active learning.

- **Detecting OOD data** Models with good calibration properties can often be applied to detecting out-of-distribution (OOD) data, or data that is significantly different from the model's training set. Although calibration and OOD detection are orthogonal problems — e.g., softmax scores can be used directly to detect OOD data despite being poorly calibrated — they are often studied in tandem, where the

best calibration methodologies are evaluated with respect to both calibration and the ability to detect OOD data (i.e., by assigning high uncertainty/low confidence to such examples)

As far as this project is concerned, it is sufficient to know the concept and the idea behind a calibrated prediction, which is why I have not dwelled on the theory of confidence calibration. In the following chapter, I will discuss the methodology used in this project and the steps that led to several experimental results.

Chapter 3

Method

3.1 Dataset

The dataset used in this work is PhysioNet MIT-BIH Arrhythmia made of labeled ECG records. (aggiungi reference da https://arxiv.org/pdf/1805.00794.pdf) The time series that compose this dataset are multivariate, but in the experiments it has been used ECG lead II re-sampled to sampling frequency of 125Hz as the input, the final data are univariate time series. The MIT-BIH dataset consists of ECG recordings from 47 different subjects recorded at the sampling rate of 360Hz. Each beat is annotated by at least two cardiologists. In accordance with Association for the Advancement of Medical Instrumentation (AAMI) EC57 standard five different beat categories have been found. The dataset consisting of five different categories of beats is strongly unbalanced, for simplicity and to avoid this problem it goes from five categories to a binary discrimination between "normal" and "abnormal" beats. All the samples are cropped, downsampled and padded with zeros to the fixed dimension of 188. This dataset consists of a series of CSV files. Each of these CSV files contain a matrix, with each row representing an example in that portion of the dataset. The final element of each row denotes the class to which that example belongs.

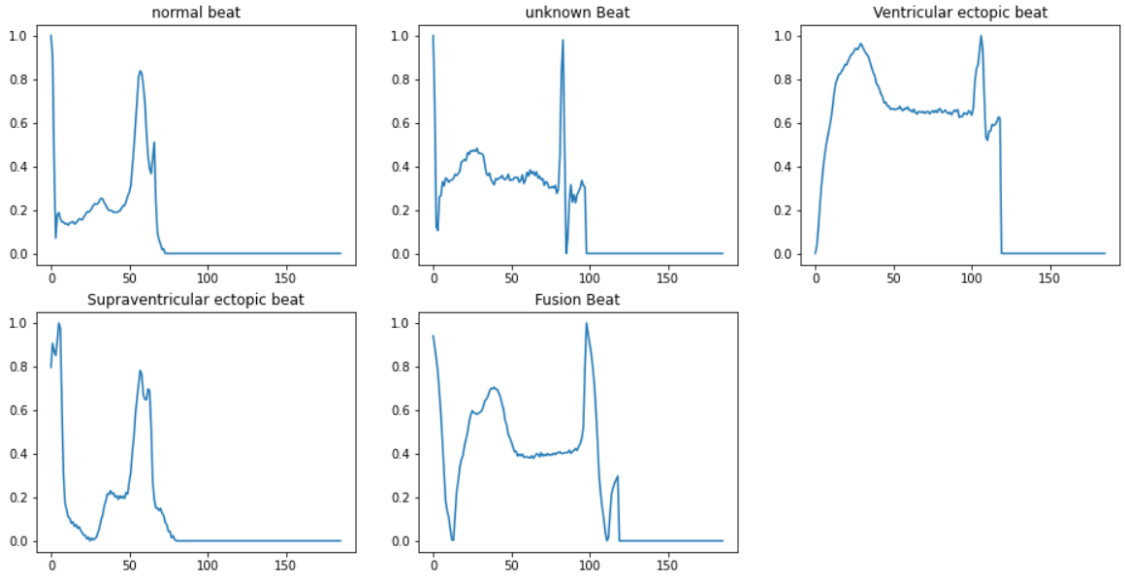
3.2 Workflow

The goal of this work is to find a correlation between an output score from an anomaly detector and the output score from a time series classifier. If this correlation exists and follows a pattern that reflects expectations, then it was thought to be able to use a class-conditioned likelihood that takes into account the anomaly score to calibrate the classifier Therefore, finally, obtain a prediction with a good degree of confidence.
The main steps that have been defined to accomplish the goal are:

1. Artificially inject noise with varying levels of noise in a well-known time series dataset;

2. Features extraction on time series using FRESH algorithm;

Figure 3.1: Univariate time series from MIT-BIH Arrythmia dataset



3. Train a classifier on the dataset made in different percentage of noisy and clean data;
4. Use a class-agnostic unsupervised method as anomaly detector, to be more specific use this method as density estimator $p(\mathbf{x})$;
5. Question: is there a relationship between the punctual loss of the classifier trained in step (3) and the learned in step (4)?;
6. If the relationship exists, model and train a new classifier which take in account the $p(\mathbf{x}|\mathbf{y})$ computed using the Bayes" rule and the density estimated at step (4).

3.2.1 Synthetic noise injection

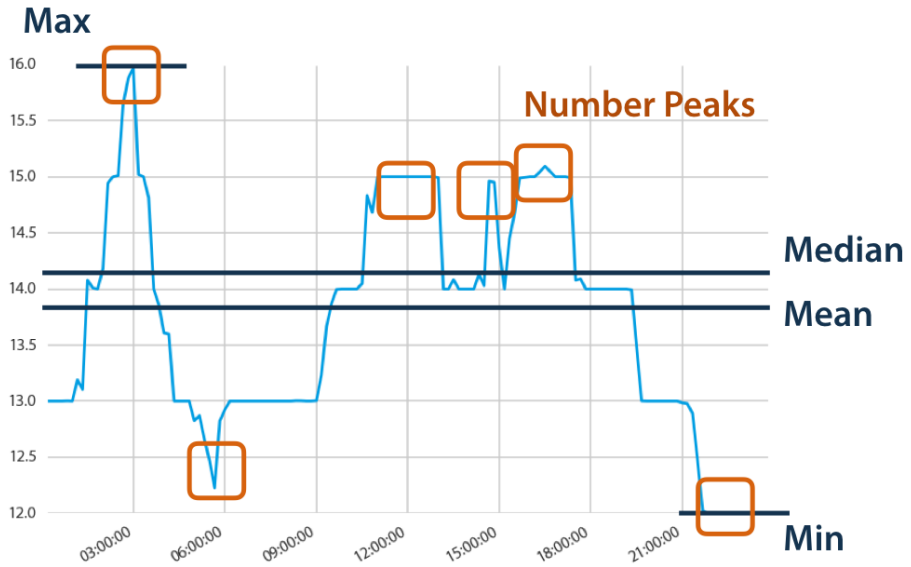
First thing, I injected artificial noise in the clean data, in detail Gaussian noise. Given n time series in the original dataset, I sample n random uniform positive numbers. For each series I apply Gaussian noise with variance equal to the number I extracted previously, in this way, some series will be very noisy (e.g. variance equal to 0.9) and others practically clean (e.g. variance equal to 0.1). The outputs of this step are two time series datasets with same dimension, one is clean and the other one is contaminated.

3.2.2 Feature extraction

This phase takes as inputs the outputs of the previous stage. On both of the datasets is applied a feature extraction algorithm, called FRESH, explained in detail in Chapter 2. The Python package tsfresh supports this process by providing automated time series feature extraction and selection on basis of the FRESH algorithm. Library tsfresh implements the application programming interfaces of the most popular Python machine

learning and data analysis frameworks such as scikit-learn, numpy, pandas, scipy, keras or tensorflow. This enables users to seamlessly integrate tsfresh into complex machine learning systems that rely on state-of-the-art Python data analysis packages. The `feature extraction` submodule contains both the collection of feature calculators and the logic to apply them efficiently to the time series data. The main public function of this submodule is `extract features`. The number and parameters of all extracted features are controlled by a settings dictionary. The `feature selection` submodule provides the function `select features`, which implements the highly parallel feature selection algorithm. The initial number of features extracted was 217 then through a filtering phase, the final number was 202, still to many. For sake of simplicity, I decided to manually choose 35 final features which were used in the most important phases of research. Those features describe basic characteristics of the time series such as the number of peaks, the average or maximal value or more complex features such as the time reversal symmetry statistic. These are the features extracted:

Figure 3.2: Features extraction on time series



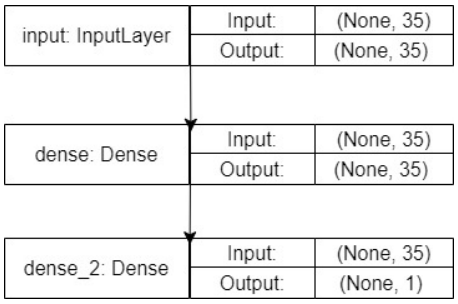
- sum values
- median
- length
- standard deviation
- variance
- root mean square
- maximum

- absolute maximum
- minimum
- abs energy
- benford correlation
- count above mean
- first location of maximum
- first location of minimum
- longest strike above mean
- variation coefficient
- sample entropy
- binner entropy max bins 10
- fourier entropy bins 2
- fourier entropy bins 3
- fourier entropy bins 5
- fourier entropy bins 10
- fourier entropy bins 100
- number cwt peaks n 1
- number cwt peaks n 5
- number peaks n 1
- number peaks n 3
- number peaks n 5
- number peaks n 10
- number peaks n 50
- percentage of reoccurring values to all values
- skewness

3.2.3 Classifier training

The classifier goal is a binary task that aim to distinguish between normal and abnormal time series, but instead of taking the time series raw data as input, it gets the features extracted from the original time series. The MLP used in this experiments is pretty simple and a graphical representation is shown in the figure below.

Figure 3.3: MLP Architecture



It is made of one hidden layer, with the same number of neurons of the input layers. In all experiments, TensorFlow computational library is used for model training and evaluation. Cross entropy loss on the softmax outputs is used as the loss function. For training the networks, it is used Adam optimization method and early stopping.
The classifier is trained and tested on mixed data, that is, data in different percentages contaminated and clean. In order to do a meaningful trend analysis, I tried different percentages as seen in the table below.

Clean % - Contaminated %	Accuracy	AUC Score
(90 - 10)	0.9213	0.9640
(80 - 20)	0.9134	0.9615
(70 - 30)	0.9070	0.9551
(60 - 40)	0.8891	0.9514

Table 3.1: Classifier results when trained and tested on mixed data

In order to observe thoroughly the classifier trend. I tested the classifier on all contaminated test set, results in the table below.

Clean % - Contaminated %	Accuracy test set all contaminted
(80 - 20)	0.7220
(70 - 30)	0.6524
(60 - 40)	0.6416

Table 3.2: Classifier results when trained on mixed data and test on all contaminated data

3.2.4 Anomaly detection

This is a delicate phase of the project. This phase can be logically divided into two sub-phase. The first one regards anomaly detection using unsupervised methods, as already mentioned in this project were developed two different architecture to detect anomalies, Gaussian Mixture Model and Variational Autoencoder. The second one aim to find a correlation between the score of the classifier trained in the previous stage and the one coming as output from the detection method. The correlation/relationship between these two elements makes it possible to compute the likelihood on the whole dataset and to use the likelihood in the training phase of the classifier to obtain a better calibrated prediction.

Anomaly detection with Gaussian Mixture Model

Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. A Gaussian Mixture is a function that is comprised of several Gaussians, each identified by $k \in 1, \ldots, K$, where K is the number of clusters in the dataset.
Before even implementing the model I decided to determine the best number k of cluster in the dataset using two popular methods: Elbow method and Silhouette method. The elbow method plots the value of the cost function produced by different values of k. If k increases, average distortion will decrease, each cluster will have fewer constituent instances, and the instances will be closer to their respective centroids. However, the improvements in average distortion will decline as k increases. The value of k at which improvement in distortion declines the most is called the elbow, at which we should stop dividing the data into further clusters.
The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from -1 to $+1$, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighbouring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.
From both elbow and silhouette method it turned out to be $k = 3$ the best number of cluster given the dataset.
Once known the k, Gaussian mixture model is implemented using scikit-learn. Also in this case, the model is trained and tested first with mixed data, later trained with mixed data and tested with only contaminated data. The results from GMM comes from the method `score_samples(X)` where X is the test set for which I want to compute the score. This method compute the log-likelihood of each sample. What is expected, then, is that poorly dirty and clean data will have close log-likelihood values, while very dirty data will have significantly lower log-likelihood values.
Once the score from the GMM are obtained and once the classifier is trained on the same data the GMM is trained. I compute the *Punctual Loss* from the classifier, to be more specific, for each test sample I compute the loss coming from the classifier. As a final step, for each contaminated data, I divide the data into bins with contamination ranging from 0 to 0.1, then from 0.1 to 0.2, and so on up to 0.9. For each of these bins, I calculate

the average of the score coming out of the GMM and the punctual loss coming out of the classifier and evaluate the correlation. The results will be shown in the next chapter.

Anomaly detection with Variational Autoencoder

An autoencoder is a deep learning model that is usually based on two main components: an encoder that learns a lower-dimensional representation of input data, and a decoder that tries to reproduce the input data in its original dimension using the lower-dimensional representation generated by the encoder. An autoencoder learns to encode the input data by trying to minimize the reproduction error or the difference between the original input vector and the output vector that was reproduced by the decoder from encoded data. If the autoencoder is sufficiently trained to produce a good reproduction of the input data it was trained on, and assuming that it was trained on enough data, then it supposes to produce a more or less stable and minimal reproduction error when it is fed with data that is "similar" to the data that it was trained on. However, it also means that an unusual or extreme reproduction error probably means that the AE has encountered an input vector that is very different than the input it was trained on and therefore it failed to properly reproduce it. If the data shown to our AE is supposed to be similar to the data it was trained on then an input that generates an extreme reproduction error is likely to be an anomaly.
In a Variational Autoencoder (VAE), the encoder similarly learns a function that takes as its input a vector of size n. However, a VAE learns to generate two vectors (of size m) that represent the parameters (mean and variance) of a distribution from which the latent vector is sampled, and which the decoder function can transform back to the original input vector. Simply put, the VAE's learning task is to learn a function that will generate parameters of distributions from which a latent vector that a decoder can easily reproduce can be sampled.
The purpose of this part is to quickly delve into the implementation code of a VAE that can detect anomalies. I followed the Keras Variational Autoencoder documentation.

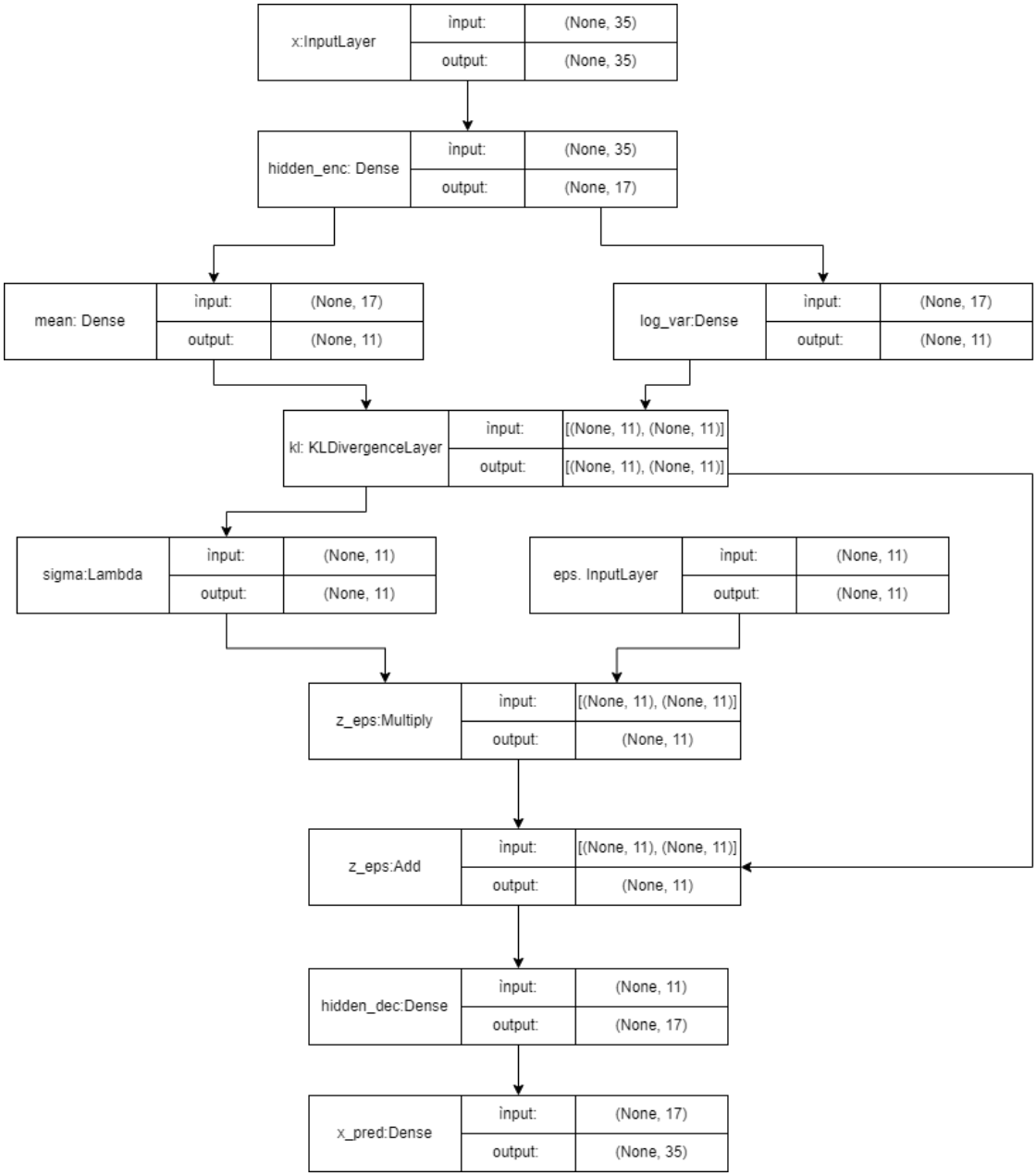
1. **Encoder**: takes as an input a vector of size n and generates the latent vector (z). The encoder first learns the mean and (log) variance of the distribution of z (i.e., z_mean and z_log_var respectively). Then it samples z from this distribution using a lambda layer calling the function `sample(`z_mean`, `z_log_var`)`. The purpose of the `sample` function is to sample the normally distributed z by returning its $mean + \sigma * \epsilon$. The role of ϵ is to ensure te continuity of the latent space.

2. **Decoder**: it takes the sampled latent vector z as its input and tries to reproduce it (generative component).

Reconstruction approaches to anomaly detection identify anomalies by their relatively high reconstruction error. Therefore, these methods work best when the model can be first trained on normal, or mostly normal, data. In this way, it can be increased the confidence that a relatively high reconstruction error was caused by a genuine anomaly.

In the next chapter, I will comment on the results of this phase. The emerging observations are the central part of my project.

Figure 3.4: Variational Autoencoder architecture



3.2.5 Class-conditioned likelihood estimation

For this step to apply, there must be a significant correlation in the previous step. Once the correlation found at the previous step is defined as meaningful, from the implemented anomaly detection models, the $p(x)$ of the data is extracted. In other words, both the GMM and VAE are used as density estimator. Given the $p(x)$ and the $p(y|x)$ (this pdf is the output of the classifier). From the Bayes' rule $p(x|y) = \frac{p(x,y)}{p(y)} = \frac{p(y|x)}{p(y)}$, where $p(y)$ is

given by dividing the number of samples in the classes by its cardinality. This quantity should tell us how "representative" of class y the point x is.

3.2.6 Classifier re-training

This last step takes into account the $p(x|y)$ computed in the previous step. The main idea is to model and train a new classifier which include $p(x|y)$, two are the paths to take.

1. Overweight points with low $p(x|y)$, these points catch edge cases in the classification problem.
2. Discard points with low $p(x|y)$ to denoise the classifier.

Chapter 4

Results

4.1 First Attempt: Gaussian Mixture Model on features extracted from time series

In this section, I present the first round of experiments and the following results. The datasets used in this section are the two features extracted dataset.

- time series raw data → feature extraction → dataset clean
- time series raw data → noise injection → feature extraction → dataset contaminated

The anomaly detector implemented is a Gaussian Mixture Model. Each section refers to different percentages of data splitting, to be more clear the first percentage value always refers to the percentage of clean data in the dataset while the second percentage value refers to the percentage of contaminated data in the dataset. For example, 90% of the data comes from the clean dataset and 10% of the data comes from the contaminated dataset.

4.1.1 Mix 90-10

Classifier results

The classifier architecture is explained and presented in Chapter 3. The main information and results are:

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 33
- Accuracy on test set: 0.9213
- AUC Score: 0.9640

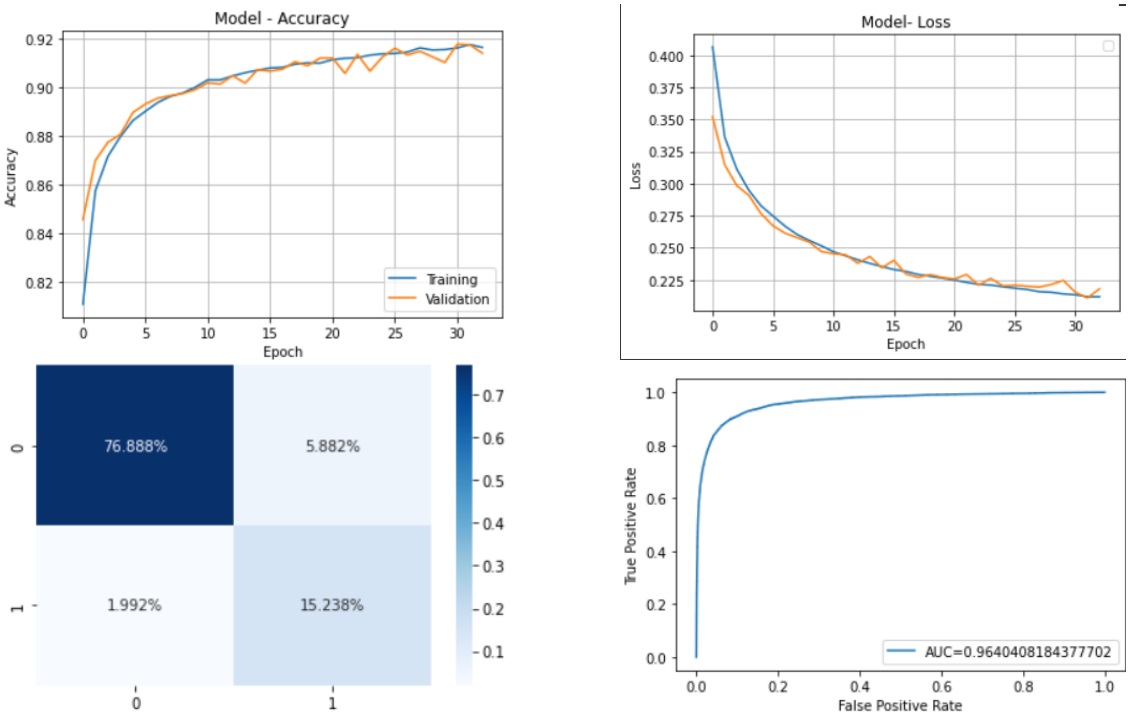


Figure 4.1: Model 90-10 results

Correlation results

In this section, I consider the contaminated data and their noise parameters. I divide the data into bins with contamination ranging from 0 to 0.1, then from 0.1 to 0.2, and so on up to 0.9. For each of these bins, I compute the mean of the score coming out of the GMM and the punctual loss coming out of the classifier and evaluate the correlation. Then, I compute standard deviation. For both mean and standard deviation I also computed a confidence interval with bootstrap.

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0, 0.1]	-15.627780	0.339448	447
(0.1, 0.2]	-11.562050	0.458804	411
(0.2, 0.3]	-8.907404	0.423618	430
(0.3, 0.4]	-8.281731	0.519411	406
(0.4, 0.5]	-12.251381	0.538863	486
(0.5, 0.6]	-12.474466	0.481026	444
(0.6, 0.7]	-16.508182	0.516615	448
(0.7, 0.8]	-21.235824	0.569505	403
(0.8, 0.9]	-21.235824	0.569505	403

Table 4.1: Mean correlation table 90-10

4.1.2 Mix 80-20

The experiments done in this mix are different from the previous one, in this case I decided to test the classifier and the GMM on all contaminated test set.

Classifier results

The main information and results are:

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 28
- Accuracy on test set: 0.7220
- AUC Score: 0.9615

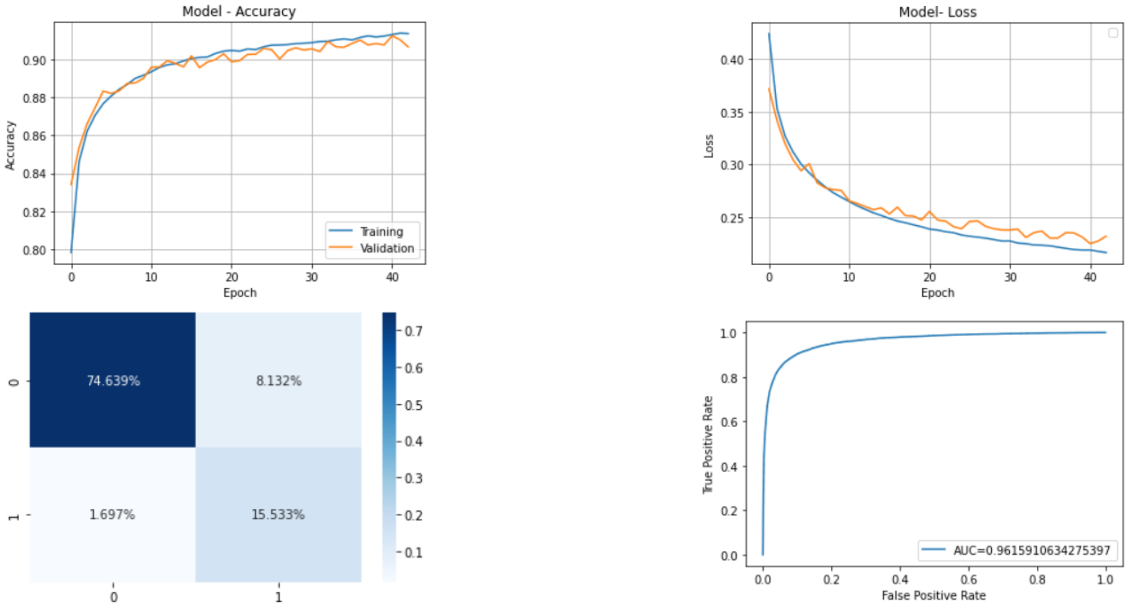


Figure 4.2: Model 80-20 results

Correlation results

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0, 0.1]	-18.948928	0.539944	4469
(0.1, 0.2]	-14.536341	0.674554	4428
(0.2, 0.3]	-12.969266	0.643553	4315
(0.3, 0.4]	-12.976513	0.632151	4271
(0.4, 0.5]	-14.246201	0.653782	4390
(0.5, 0.6]	-15.283476	0.678828	4391
(0.6, 0.7]	-16.757100	0.728251	4447
(0.7, 0.8]	-18.490283	0.767304	4329
(0.8, 0.9]	-22.255502	0.799899	4379

Table 4.2: Mean correlation table on 80-20 test set all contaminated

4.1.3 Mix 70-30

Classifier results

The classifier architecture is explained and presented in Chapter 3. The main information and results are:

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 19
- Accuracy on test set: 0.9070
- AUC Score: 0.9551

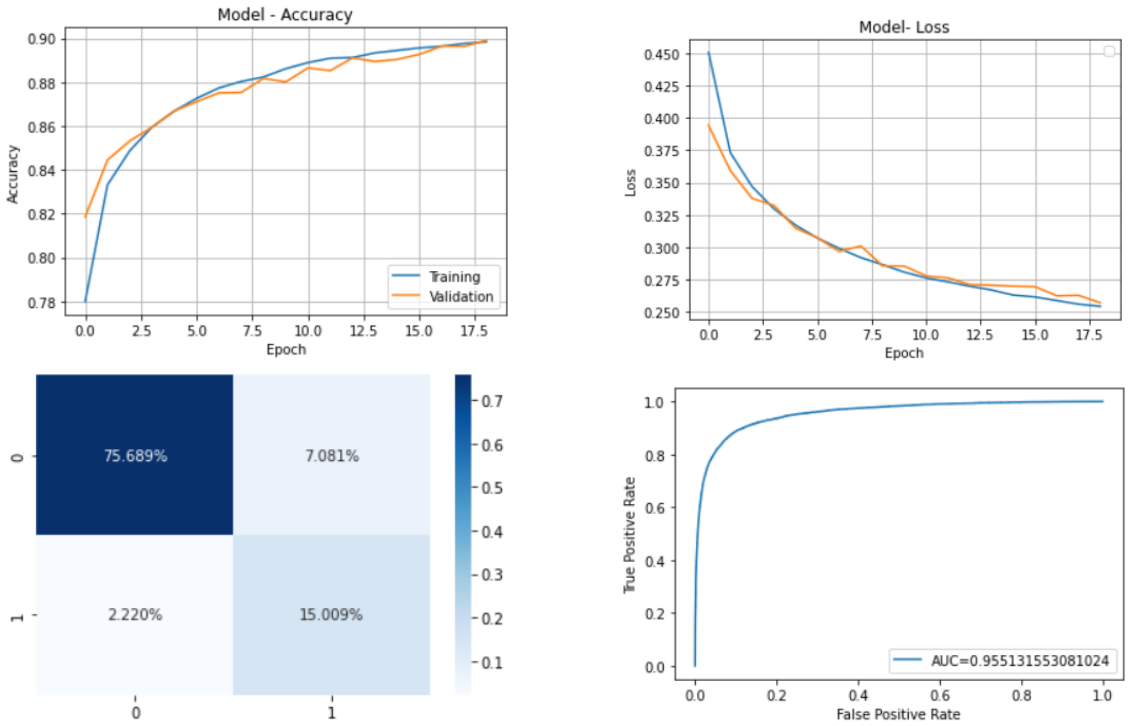


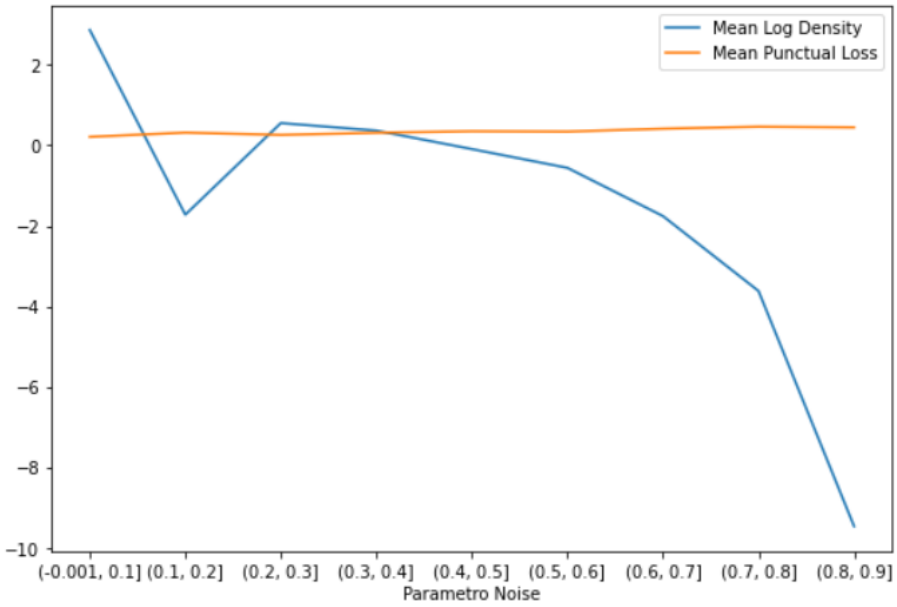
Figure 4.3: Model 70-30 results

Correlation results

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0, 0.1]	-2.393209	0.301150	1323
(0.1, 0.2]	-1.722611	0.315670	1316
(0.2, 0.3]	0.552621	0.258184	1291
(0.3, 0.4]	0.364981	0.311405	1289
(0.4, 0.5]	-0.093850	0.349352	1325
(0.5, 0.6]	-0.560022	0.341285	1307
(0.6, 0.7]	-1.758111	0.412344	1345
(0.7, 0.8]	-3.617692	0.462201	1274
(0.8, 0.9]	-9.459266	0.445434	1280

Table 4.3: Correlation 70-30

Figure 4.4: Plot of correlation 70-30



Classifier results with test set all contaminated

The main information and results are:

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 22
- Accuracy on test set: 0.6524

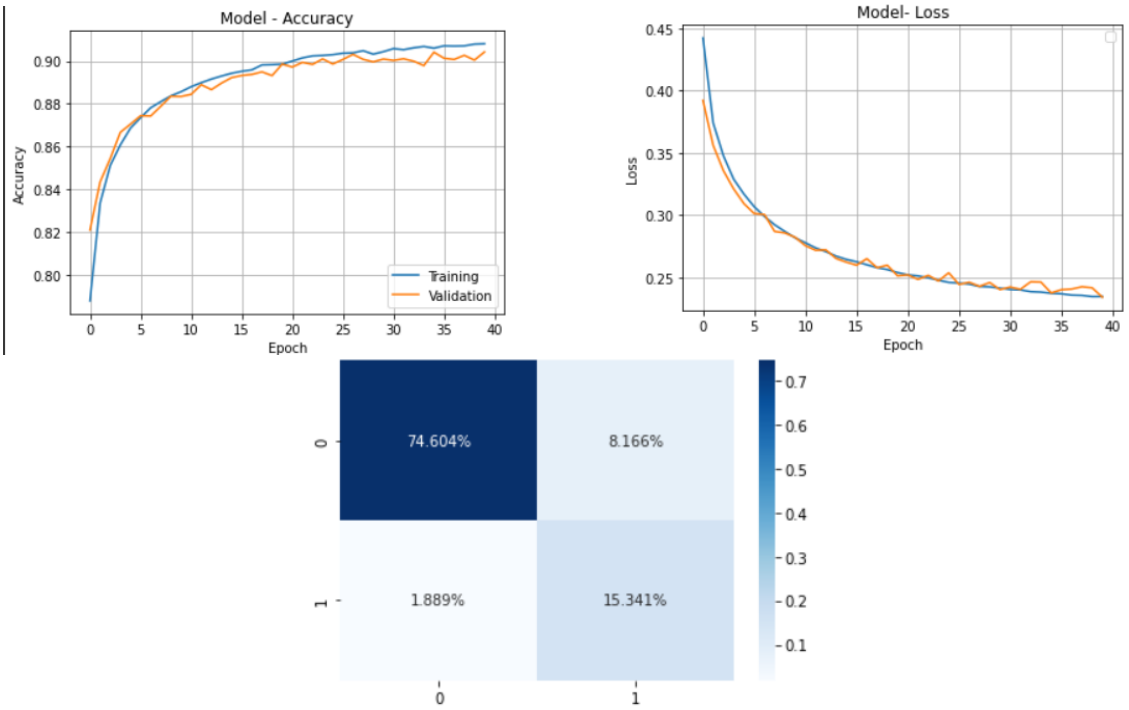


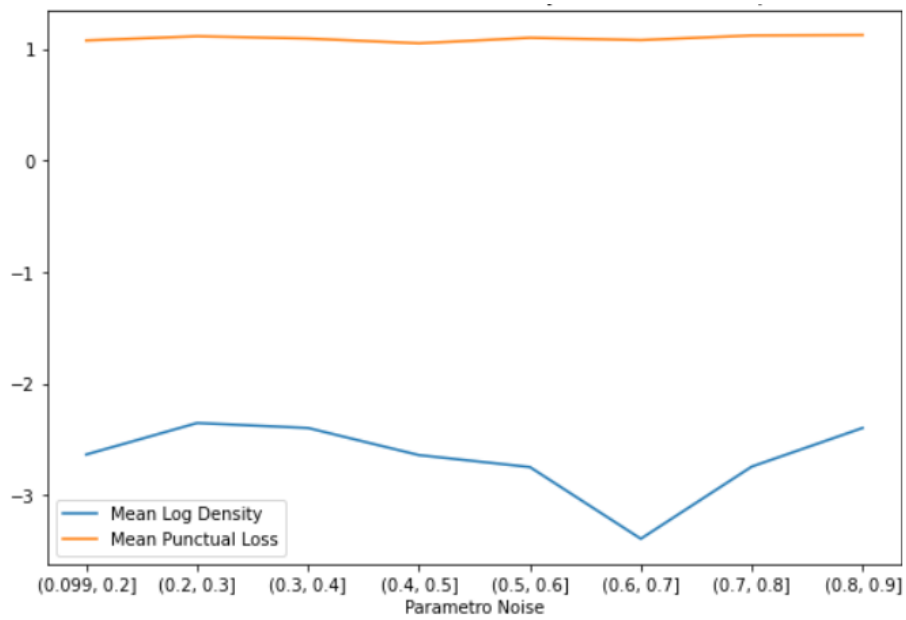
Figure 4.5: Model 70-30 results

Correlation results with test set all contaminated

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0, 0.1]	-2.841234	1.121904	4472
(0.1, 0.2]	-2.638512	1.078769	4327
(0.2, 0.3]	-2.355713	1.116544	4426
(0.3, 0.4]	-2.400468	1.095751	4336
(0.4, 0.5]	-2.643988	1.053106	4402
(0.5, 0.6]	-2.751298	1.102526	4341
(0.6, 0.7]	-3.395038	1.082285	4319
(0.7, 0.8]	-2.747146	1.123283	4337
(0.8, 0.9]	-2.399955	1.127237	4398

Table 4.4: Correlation test set all contaminated

Figure 4.6: Plot of correlation 70-30 test set all contaminated



4.1.4 Mix 60-40

Classifier results

The main information and results are:

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 27
- Accuracy on test set: 0.8891
- AUC Score: 0.9514

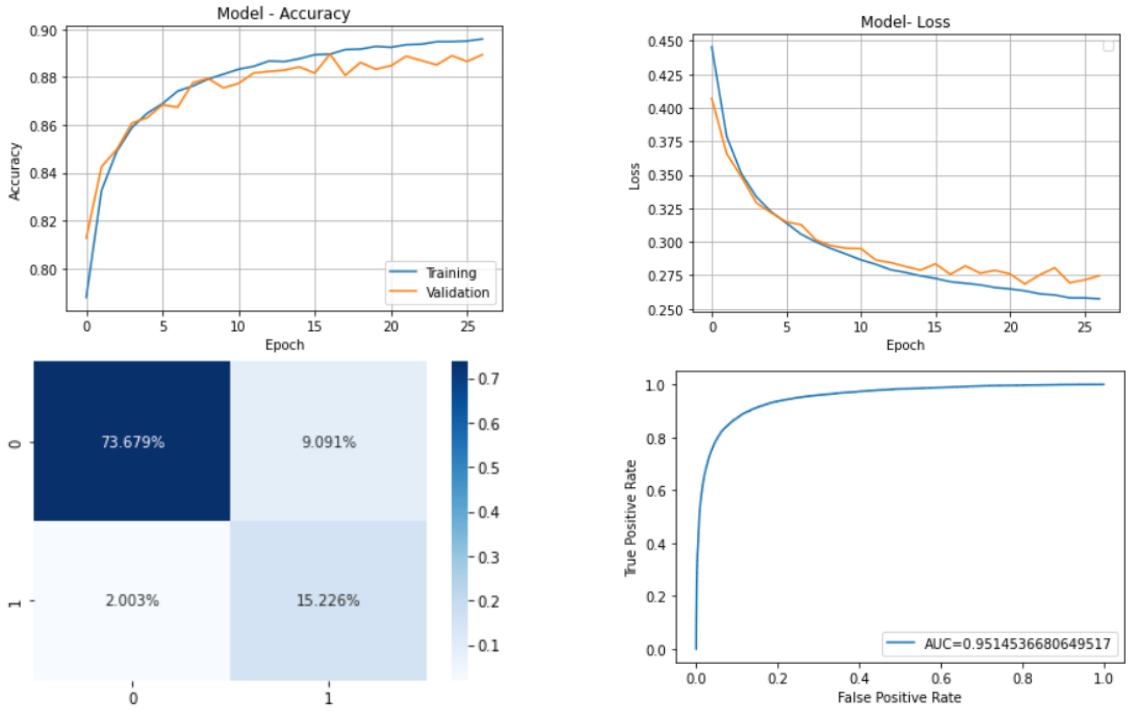


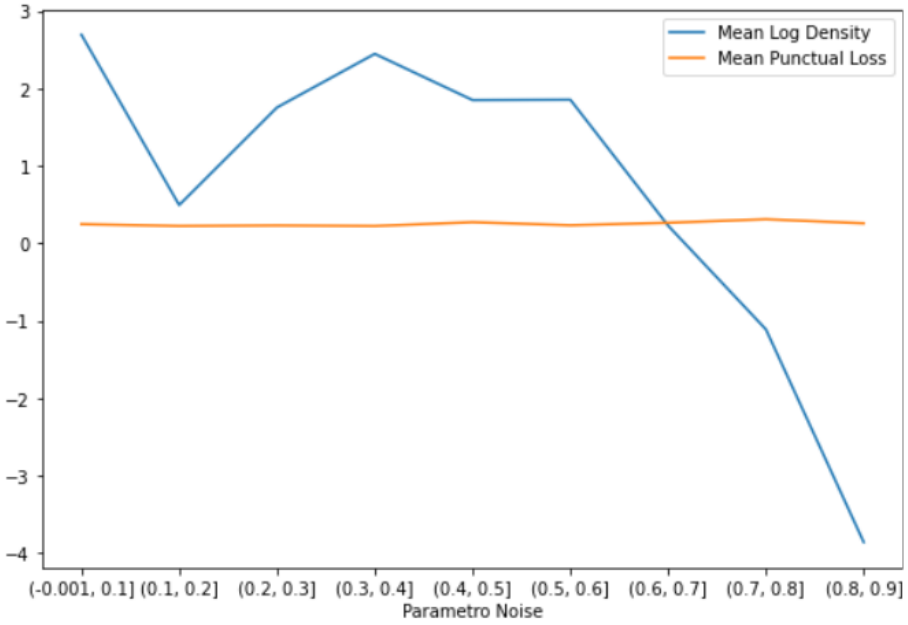
Figure 4.7: Model 60-40 results

Correlation results

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0, 0.1]	-0.853594	0.245896	1836
(0.1, 0.2]	0.496548	0.234424	1812
(0.2, 0.3]	1.757241	0.234539	1746
(0.3, 0.4]	2.450092	0.244296	1713
(0.4, 0.5]	1.853068	0.301424	1769
(0.5, 0.6]	1.857895	0.295378	1717
(0.6, 0.7]	0.230649	0.328197	1711
(0.7, 0.8]	-1.108893	0.362332	1684
(0.8, 0.9]	-3.857562	0.353005	1714

Table 4.5: Correlation 60-40

Figure 4.8: Plot of correlation 60-40



Classifier results with test set all contaminated

The main information and results are:

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 22
- Accuracy on test set: 0.6416

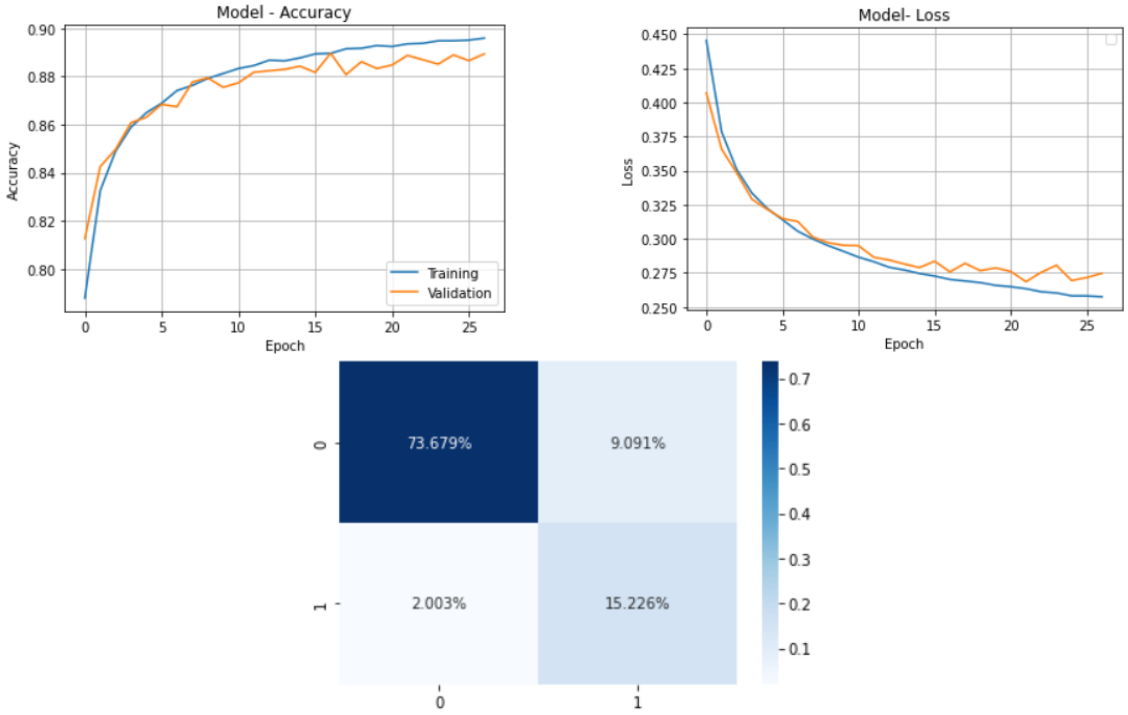


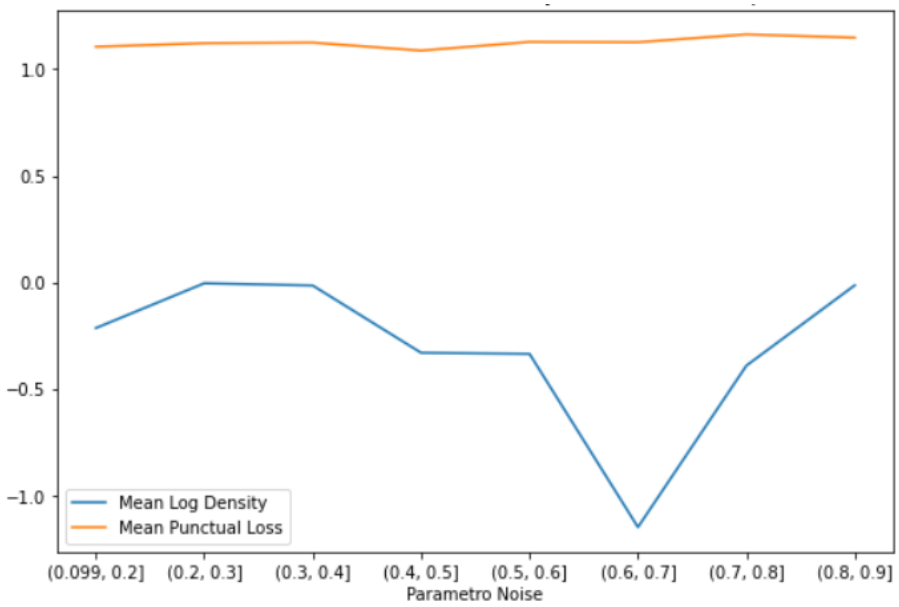
Figure 4.9: Model 60-40 results test set all contaminated

Correlation results with test set all contaminated

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0, 0.1]	-0.447251	1.156162	4472
(0.1, 0.2]	-0.214423	1.105412	4327
(0.2, 0.3]	-0.004207	1.121949	4426
(0.3, 0.4]	-0.014667	1.124722	4336
(0.4, 0.5]	-0.330368	1.087210	4402
(0.5, 0.6]	-0.335292	1.127865	4341
(0.6, 0.7]	-1.148233	1.126519	4319
(0.7, 0.8]	-0.390617	1.163275	4337
(0.8, 0.9]	-0.013459	1.147852	4398

Table 4.6: Correlation 60-40 test set all contaminated

Figure 4.10: Plot of correlation 60-40 test set all contaminated



4.1.5 Observations on first attempt

In this section, I will assess the outcomes of the initial experiments conducted on observations. Specifically, I will examine the trend that emerges when the attempts are evaluated on both fully contaminated and mixed data.

The first point I want to mention concerns the loss values at specific points. The loss shows a predictable pattern where, as the model is tested with both clean and dirty data, the loss values remain around zero and tend to increase with higher levels of noise in the data. In the case of a test set that is entirely contaminated, the loss values are around one and tend to increase with higher levels of noise in the data.

However, this is not the case with the log density at the GMM output. We expected the log density values to decrease as the noise level increased. Instead, the log density tends to decrease in cases of intermediate noise, but it remains the same for low or high levels of noise. The last point I would like to make, concerns a comparison of the log density trend, with respect to an all contaminated test and a mixed test set. In the case of the mixed set, the results on very high noise have a clear downward trend. In the case of all contaminated, there is still a U-shaped trend, but it is not as marked as in the previous case and the downward trend is not present.

4.2 Second Attempt: Noise injection on features extracted

In this section, I do a check given the observations referred to the results of the first attempt. Instead of performing noise injection on the time series, I perform noise injection directly on the features extracted from the time series. The datasets used are two.

- time series raw data → features extraction → dataset clean
- time series raw data → features extraction → noise injection → dataset anomalized

4.2.1 Mix 80-20

Classifier results

The main information and results are:

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 18
- Accuracy on test set: 0.9017
- AUC Score: 0.9618

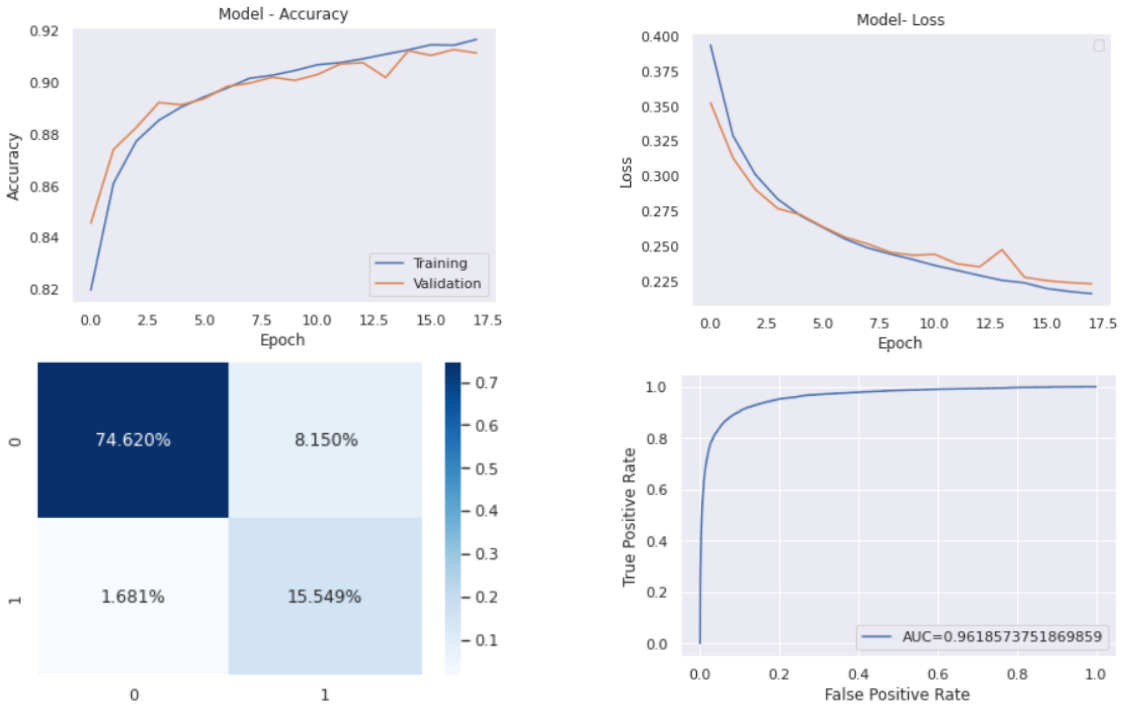


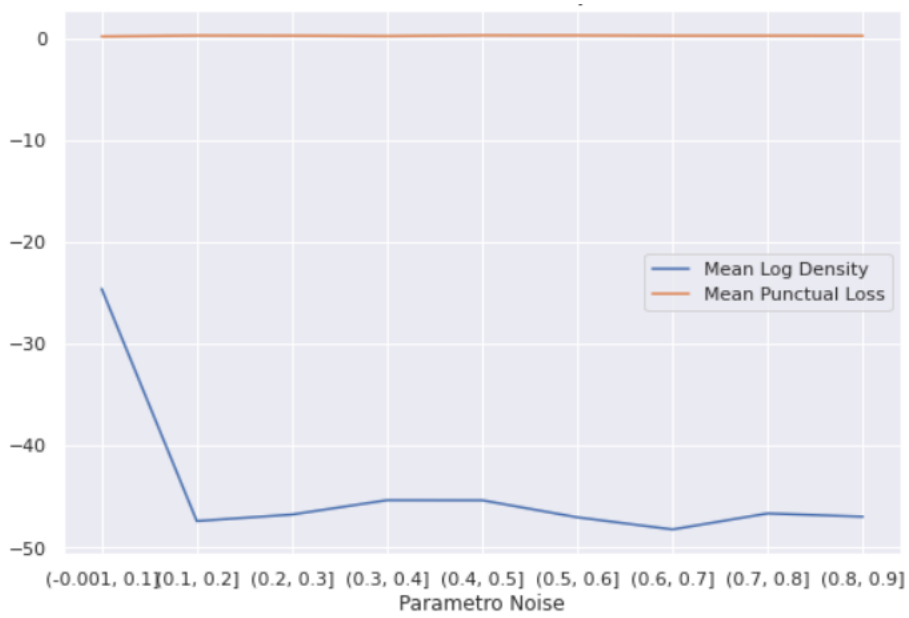
Figure 4.11: Model 80-20 results

Correlation results

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0, 0.1]	-48.559932	0.230494	842
(0.1, 0.2]	-47.379800	0.321635	886
(0.2, 0.3]	-46.738275	0.306743	891
(0.3, 0.4]	-45.327699	0.270249	875
(0.4, 0.5]	-45.339208	0.342461	909
(0.5, 0.6]	-47.012512	0.335860	837
(0.6, 0.7]	-48.202892	0.306258	869
(0.7, 0.8]	-46.629199	0.304216	861
(0.8, 0.9]	-46.949256	0.293974	829

Table 4.7: Mean correlation table 80-20

Figure 4.12: Plot correlation 80-20



4.2.2 Mix 70-30

Classifier results

The main information and results are:

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 17
- Accuracy on test set: 0.9075
- AUC Score: 0.9609

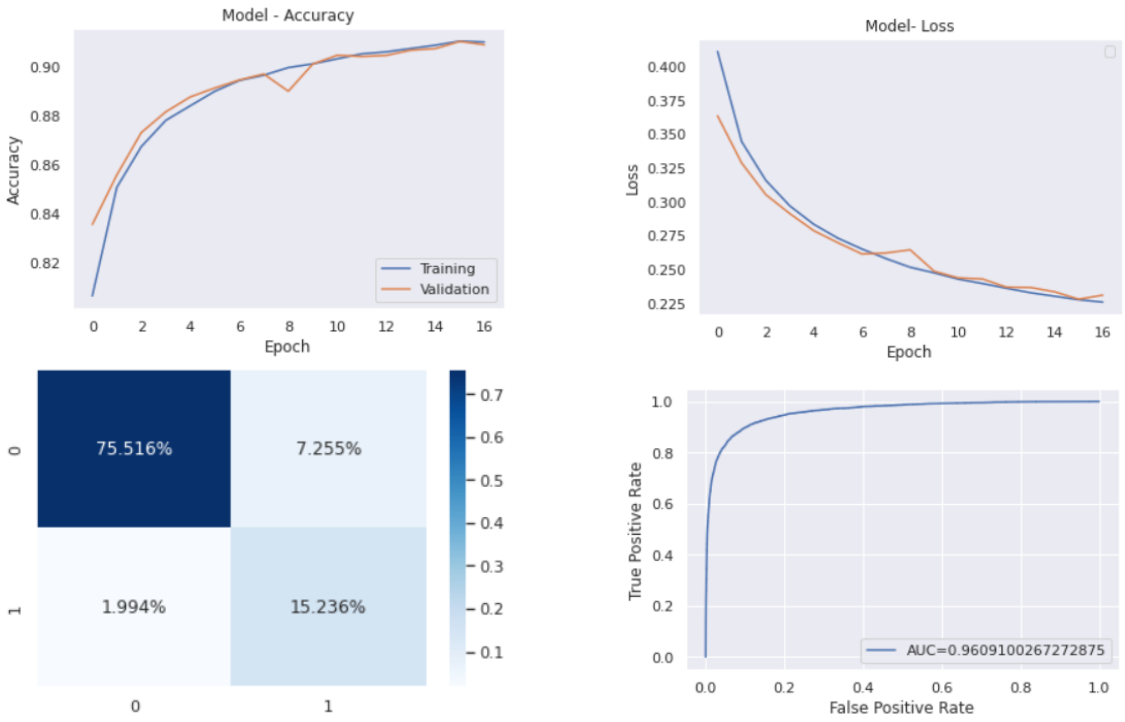


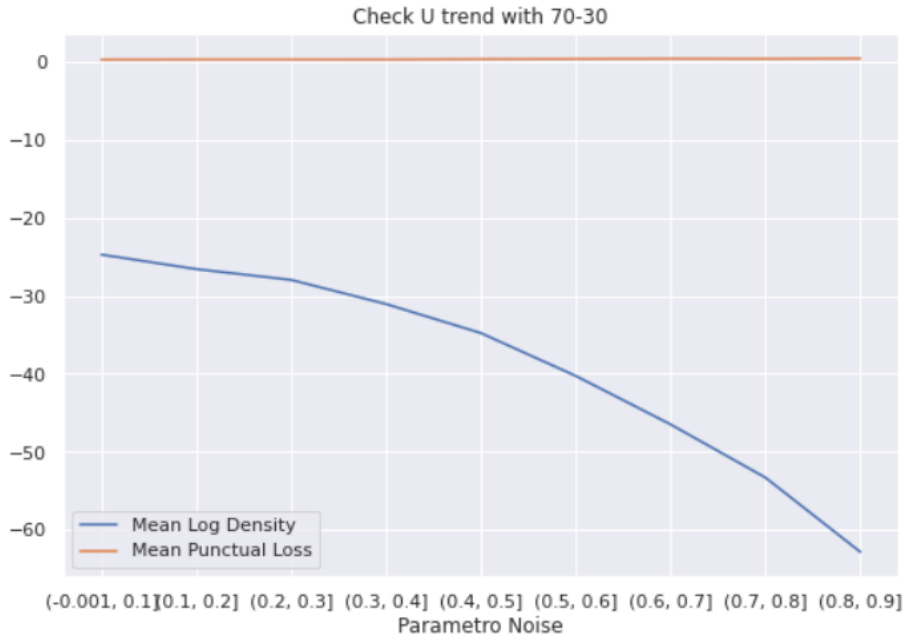
Figure 4.13: Model 70-30 results

Correlation results

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0, 0.1]	-28.739045	0.210720	1248
(0.1, 0.2]	-26.591500	0.230031	1309
(0.2, 0.3]	-27.972264	0.229284	1315
(0.3, 0.4]	-31.059644	0.224958	1348
(0.4, 0.5]	-34.771632	0.265872	1290
(0.5, 0.6]	-40.254348	0.285218	1358
(0.6, 0.7]	-46.459532	0.325868	1264
(0.7, 0.8]	-53.264157	0.307126	1277
(0.8, 0.9]	-62.775841	0.350487	1259

Table 4.8: Mean correlation table 70-30

Figure 4.14: Plot correlation 70-30



4.2.3 Mix 60-40

Classifier results

The main information and results are:

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 17
- Accuracy on test set: 0.9075
- AUC Score: 0.9609

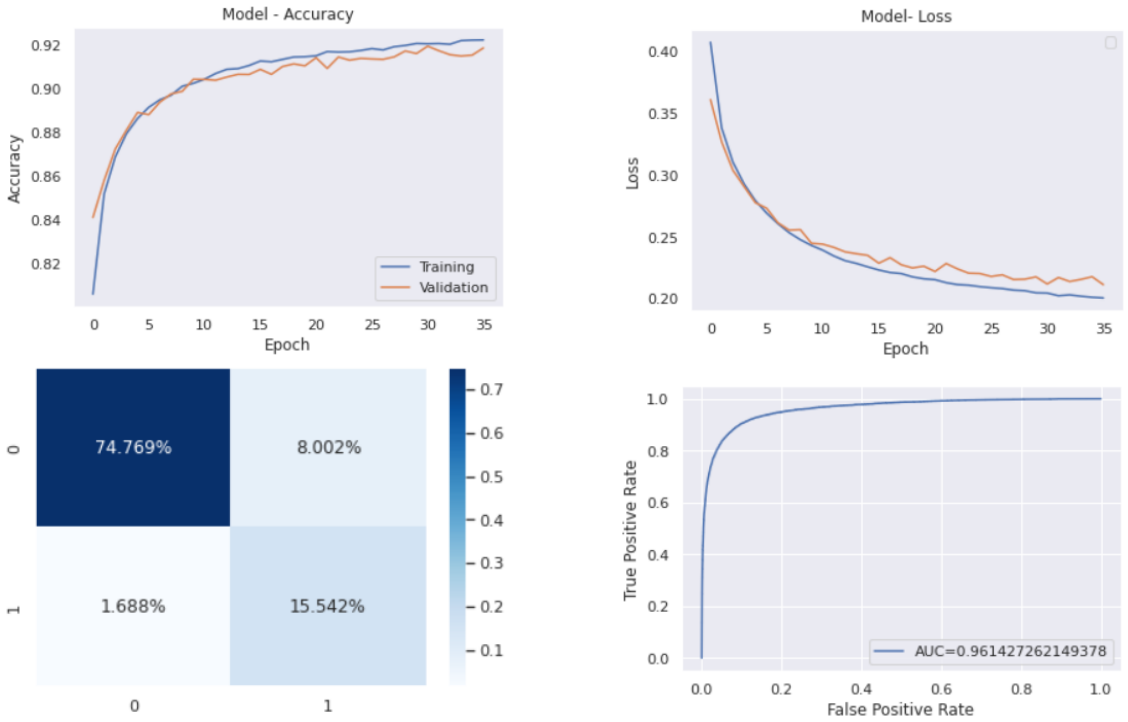


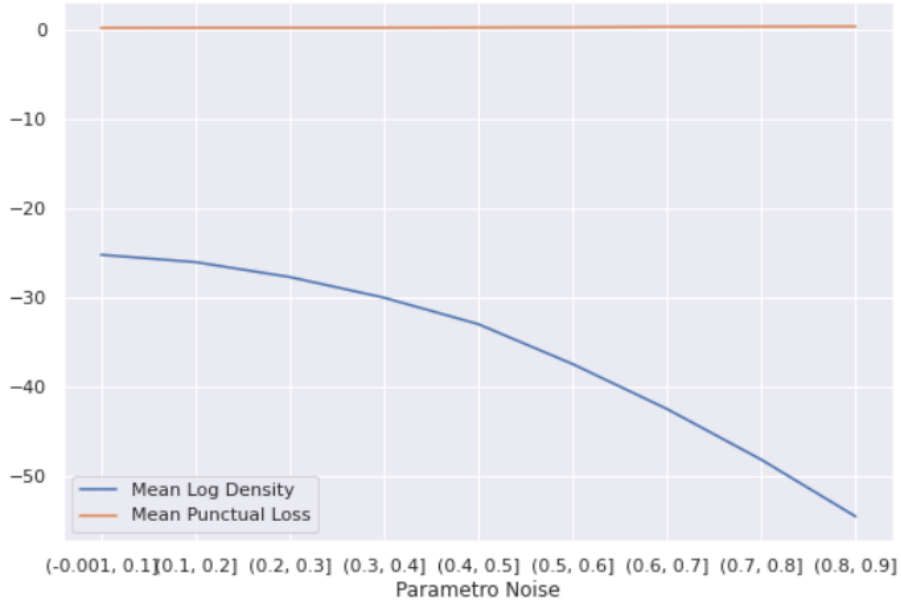
Figure 4.15: Model 60-40 results

Correlation results

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0 0.1]	-27.244678	0.215563	1742
(0.1, 0.2]	-26.067839	0.232130	1744
(0.2, 0.3]	-27.740601	0.236376	1725
(0.3, 0.4]	-30.057670	0.234927	1681
(0.4, 0.5]	-33.026746	0.260770	1770
(0.5, 0.6]	-37.504192	0.274427	1788
(0.6, 0.7]	-42.542819	0.333937	1802
(0.7, 0.8]	-48.229647	0.359181	1748
(0.8, 0.9]	-54.606974	0.377085	1750

Table 4.9: Mean correlation table 60-40

Figure 4.16: Plot correlation 60-40



4.2.4 Observations on second attempt

The second experimental attempt produces results that are markedly different from the first attempt. In these cases, particularly in the 70-30 and 60-40 mixtures, the log density trend decreases as the noise parameter injected into the data increases. This phenomenon occurs because the noise is introduced after the feature extraction phase, rather than before it.
It is worth noting, though, that this trend is not as evident in the case of the 80-20 mixture as it is in the mixes mentioned earlier.
Before drawing any definitive conclusions from the previous observations, additional experiments were conducted using a Variational Autoencoder as the detector, rather than a GMM, and by performing an analysis of time series data without going through the feature extraction phase.

4.3 Third Attempt: Analysis on time series

In this round of experiments, I decided to do time series analysis without going through the feature extraction phase. A method based on deep convolutional networks for the classification of heartbeats is proposed. The time series dataset as input to the convolutional architecture is the same as that used in the previous experiments. Again, two datasets were used, one clean and one to which variable Gaussian noise was injected. This has been tested again for mixed datasets in different percentages of clean and dirty data.

All convolution layers are applying 1-D convolution through time and each have 64 kernels of size 6. We also use max pooling of size 3. The predictor network consists of three convolutional blocks followed by batch normalization. Two dense layers, all layers have ReLu as activation function, with sigmoid in the last one. In all experiments, TensorFlow computational library is used for model training and evaluation. Binary cross entropy loss on the sigmoid output is used as the loss function. For training the networks, I used Adam optimization method and early stopping.

4.3.1 Mix 90-10

Classifier results

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 5
- Accuracy on test set: 0.9799

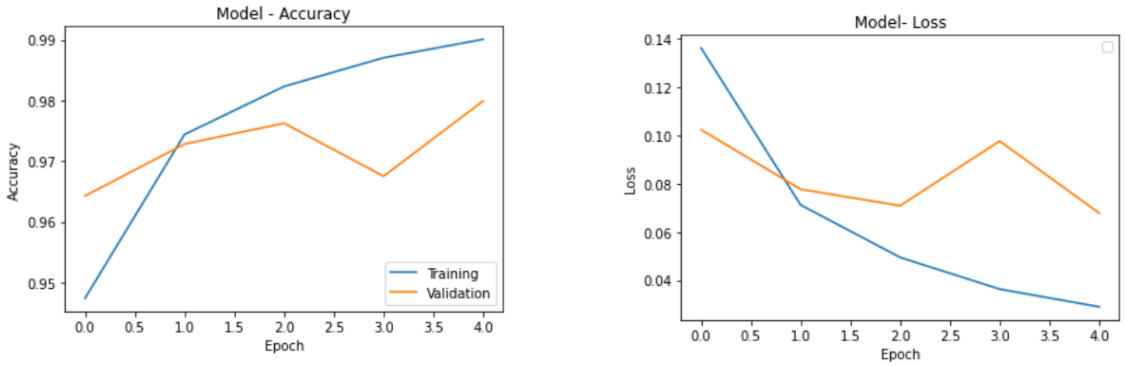


Figure 4.17: Model 90-10 results

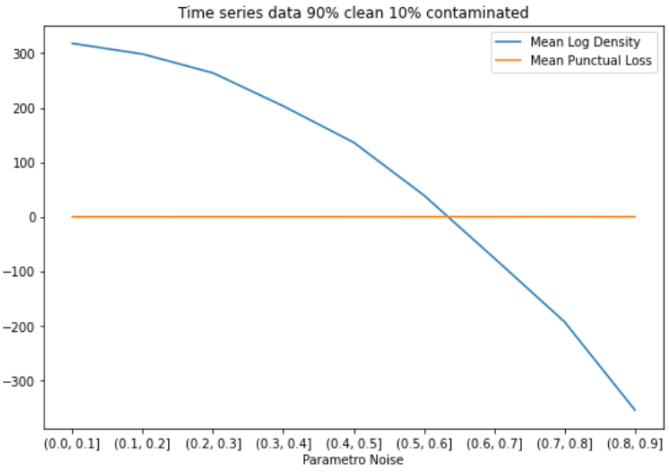
Correlation results

In this section, I consider the contaminated data and their noise parameters. I divide the data into bins with contamination ranging from 0 to 0.1, then from 0.1 to 0.2, and so on up to 0.9. For each of these bins, I compute the mean of the score coming out of the GMM and the punctual loss coming out of the classifier and evaluate the correlation.

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0, 0.1]	318.007730	0.050472	227
(0.1, 0.2]	298.450887	0.050467	217
(0.2, 0.3]	263.808622	0.024690	199
(0.3, 0.4]	203.074957	0.048809	210
(0.4, 0.5]	136.503013	0.095981	237
(0.5, 0.6]	39.523494	0.152298	226
(0.6, 0.7]	-75.725387	0.152039	235
(0.7, 0.8]	-192.623668	0.308071	202
(0.8, 0.9]	-354.187163	0.111493	224

Table 4.10: Mean correlation table 90-10

Figure 4.18: Correlation results



4.3.2 Mix 80-20

Classifier results

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 5
- Accuracy on test set: 0.9772

Correlation results

In this section, I consider the contaminated data and their noise parameters. I divide the data into bins with contamination ranging from 0 to 0.1, then from 0.1 to 0.2, and so on

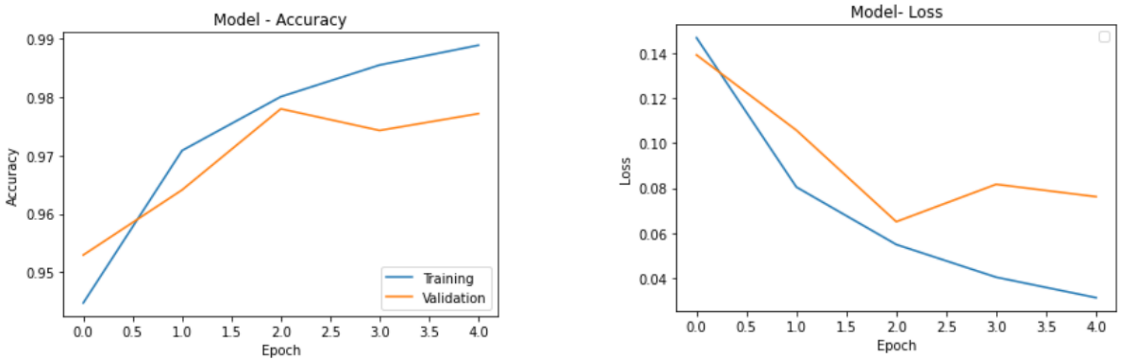


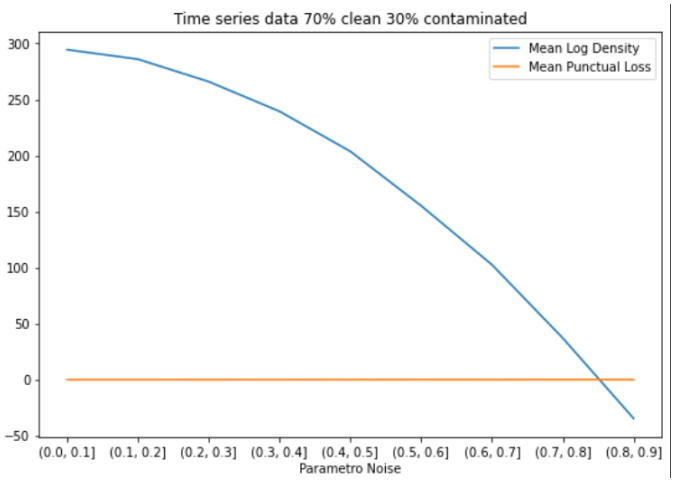
Figure 4.19: Model 80-20 results

up to 0.9. For each of these bins, I compute the mean of the score coming out of the GMM and the punctual loss coming out of the classifier and evaluate the correlation.

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0, 0.1]	303.737194	0.048911	454
(0.1, 0.2]	292.845362	0.044226	451
(0.2, 0.3]	270.008340	0.069002	403
(0.3, 0.4]	230.766708	0.039692	436
(0.4, 0.5]	183.942303	0.084793	456
(0.5, 0.6]	119.899259	0.108734	459
(0.6, 0.7]	49.224328	0.089634	466
(0.7, 0.8]	-35.868177	0.179822	439
(0.8, 0.9]	-131.165743	0.104171	450

Table 4.11: Mean correlation table 80-20

Figure 4.20: Correlation results



4.3.3 Mix 70-30

Classifier results

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 5
- Accuracy on test set: 0.9768

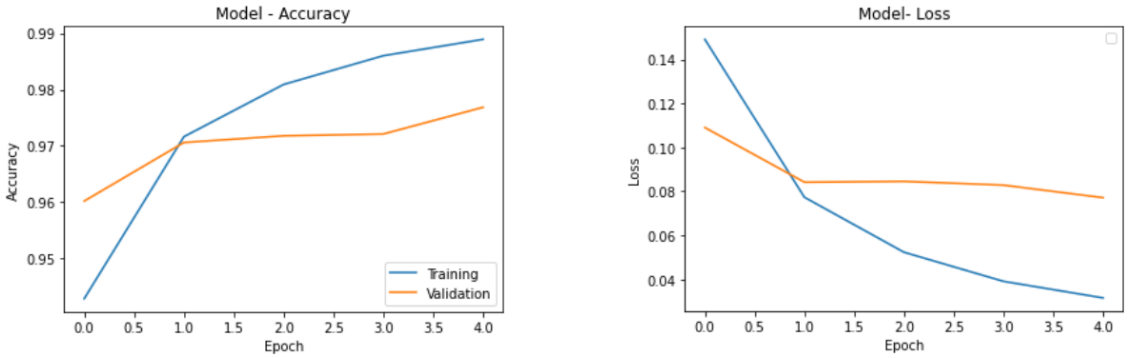


Figure 4.21: Model 70-30 results

Correlation results

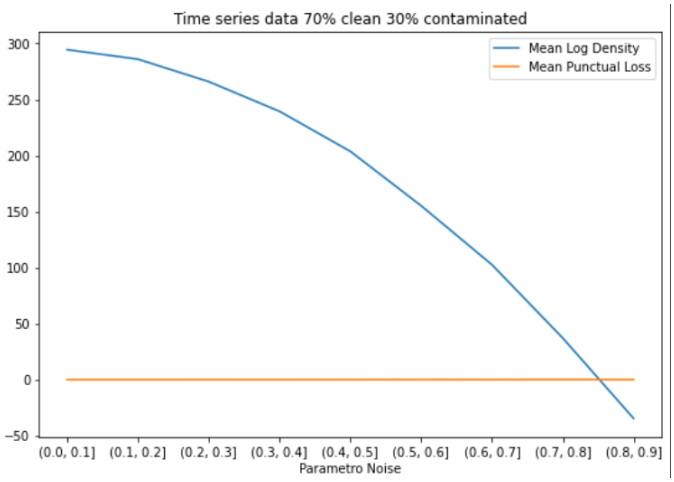
In this section, I consider the contaminated data and their noise parameters. I divide the data into bins with contamination ranging from 0 to 0.1, then from 0.1 to 0.2, and so on

up to 0.9. For each of these bins, I compute the mean of the score coming out of the GMM and the punctual loss coming out of the classifier and evaluate the correlation.

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0, 0.1]	294.440682	0.053380	689
(0.1, 0.2]	285.956653	0.047087	650
(0.2, 0.3]	265.918601	0.062139	618
(0.3, 0.4]	239.403489	0.049981	647
(0.4, 0.5]	203.697895	0.077318	700
(0.5, 0.6]	155.152608	0.109908	676
(0.6, 0.7]	102.548937	0.077213	669
(0.7, 0.8]	37.110784	0.129990	670
(0.8, 0.9]	-34.538758	0.086861	673

Table 4.12: Mean correlation table 70-30

Figure 4.22: Correlation results



4.3.4 Mix 60-40

Classifier results

- Loss: Binary Cross Entropy
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.1$ and $patience = 10$
- Number of epochs per training: 5
- Accuracy on test set: 0.9783

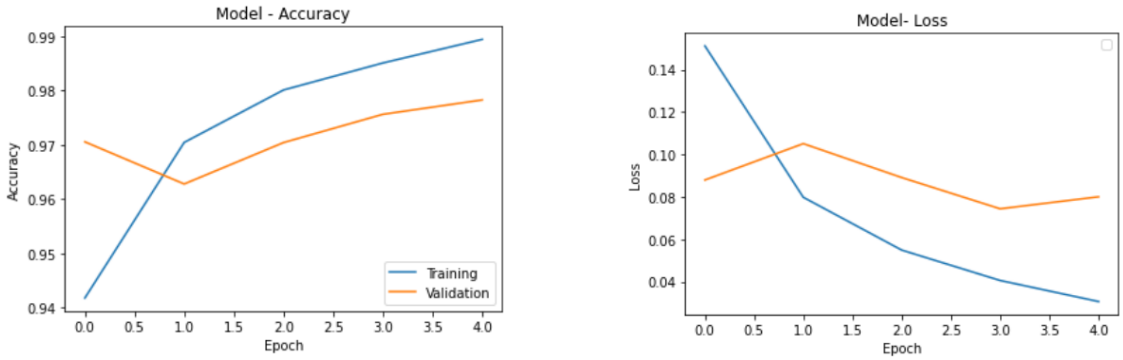


Figure 4.23: Model 60-40 results

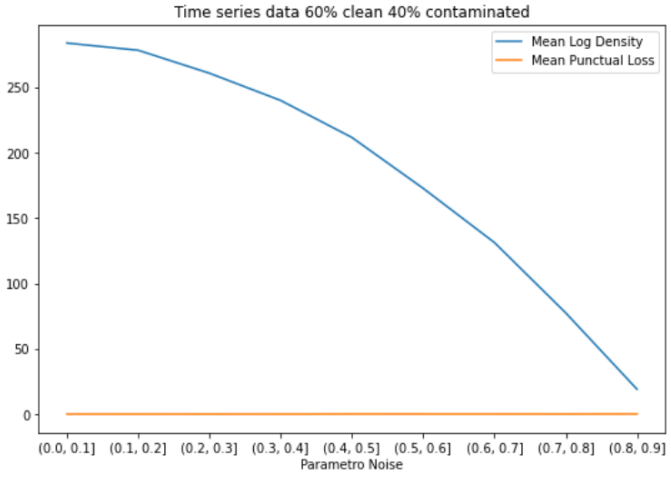
Correlation results

In this section, I consider the contaminated data and their noise parameters. I divide the data into bins with contamination ranging from 0 to 0.1, then from 0.1 to 0.2, and so on up to 0.9. For each of these bins, I compute the mean of the score coming out of the GMM and the punctual loss coming out of the classifier and evaluate the correlation.

Noise Param	Mean Log Density	Mean Punctual Loss	Total
(0.0, 0.1]	283.965353	0.059269	897
(0.1, 0.2]	278.462605	0.046527	863
(0.2, 0.3]	260.897377	0.070631	837
(0.3, 0.4]	240.038722	0.048569	835
(0.4, 0.5]	211.710451	0.121541	882
(0.5, 0.6]	172.783213	0.116231	896
(0.6, 0.7]	131.302134	0.105600	905
(0.7, 0.8]	77.508168	0.091956	895
(0.8, 0.9]	19.099828	0.153255	889

Table 4.13: Mean correlation table 60-40

Figure 4.24: Correlation results



4.3.5 Observations on third attempt

As seen from this latest round of experiments, when feature extraction is not performed, the correlation trend is exactly as expected. The log density values from the GMM tend to decrease as the noise in the data increases. When time series are analyzed using convolutional neural networks, the results are good, and the accuracy values from the classifier remain high in all cases that have been analyzed. Additionally, the loss from the classifier also tends to increase as the noise increases. Therefore, the trend observed in datasets composed of extracted features is not observed in this case where the feature extraction phase is not performed, and direct work is done on the time series. Each of the observations made in these results will be further discussed and analyzed in the last chapter where conclusions will be drawn.

4.4 Fourth Attempt: Variational Autoencoder on features extracted

The last popular approach implemented in this thesis to perform anomaly detection is based on reconstruction methods. The underlying idea is based on the assumption that if a model can learn a function that compresses and reconstructs normal data, then it will fail to do so when encountered with anomalous data because its function was only trained on normal data. The failure to reconstruct data or, more accurately, the range of the reconstruction error that it entails, can therefore signal the presence of anomalous data.

Reconstruction approaches to anomaly detection have been implemented using deep autoencoders with very good results, though an increasing body of literature suggests improved results using the more sophisticated and probablistic variational autoencoders.

4.4.1 Mix 85 - 15 with noise injection before features extraction

Model results

- Loss: Variational Autoencoder Loss (KL Loss and reconstruction loss)
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.01$ and $patience = 20$
- Number of epochs per training: 776
- Loss on last epoch: 172.1885
- Loss validation set on last epoch: 170.0732

In the figure below it is possible to compare the two distributions of data, the first one of the original data and the second one of the generated data.

Correlation results

As previously stated, generative methods can be used for detecting anomalies by leveraging the core concept that the reconstruction error for anomalous data is typically higher than that for non-anomalous data. To this end, the mean squared error between the generated data and the original data is computed. In order to ensure consistency with the results reported thus far, the mean reconstruction error was calculated for each noise level of the corrupted data in the test set.

Noise Param	Mean Error
(0.0, 0.1]	0.065346
(0.1, 0.2]	0.060148
(0.2, 0.3]	0.057578
(0.3, 0.4]	0.054135
(0.4, 0.5]	0.059281
(0.5, 0.6]	0.059040
(0.6, 0.7]	0.061373
(0.7, 0.8]	0.060711
(0.8, 0.9]	0.067574

4.4.2 Mix 80 - 20 with noise injection after features extraction

For the sake of consistency during the final stage of experimentation, I opted to train and test the identical Variational Autoencoder model using the data with injected noise after the feature extraction phase, as opposed to the previous scenario where the noise was introduced prior to this phase.

Model results

- Loss: Variational Autoencoder Loss (KL Loss and reconstruction loss)
- Optimizer: Adam
- Early Stopping on validation loss, $min\_delta = 0.01$ and $patience = 20$
- Number of epochs per training: 378
- Loss on last epoch: 73.7296
- Loss validation set on last epoch: 70.1093

Correlation results

Again, I have shown the correlation results, showing for each noise level applied to the data the average value of the reconstruction error between the generated and original data. These are the last correlation results I show before making the final observations and conclusions.

Noise Param	Mean Error
(0.0, 0.1]	0.13759
(0.1, 0.2]	0.22086
(0.2, 0.3]	0.40064
(0.3, 0.4]	0.61684
(0.4, 0.5]	0.80142
(0.5, 0.6]	1.40573
(0.6, 0.7]	1.96923
(0.7, 0.8]	2.66509
(0.8, 0.9]	3.96225

4.4.3 Observations on fourth attempt

In this final attempt, I decided to test a generative model, given its wide usage in literature. As can be seen from the correlation tables in the two different cases, in the first case, the noise is barely detected, and the value remains constant for each interval present in the tables, despite an expected increase in error with increasing noise. However, in the second case, where the model is trained and tested on data that has been corrupted after the feature extraction phase, the reconstruction error increases as the noise parameter increases. This final experimental phase allowed me to close the circle of necessary confirmations in order to draw conclusions from the results obtained. In the next section, I will go into more detail about the conclusive observations, and then conclude with the final section on future improvements related to the topic addressed in this thesis.

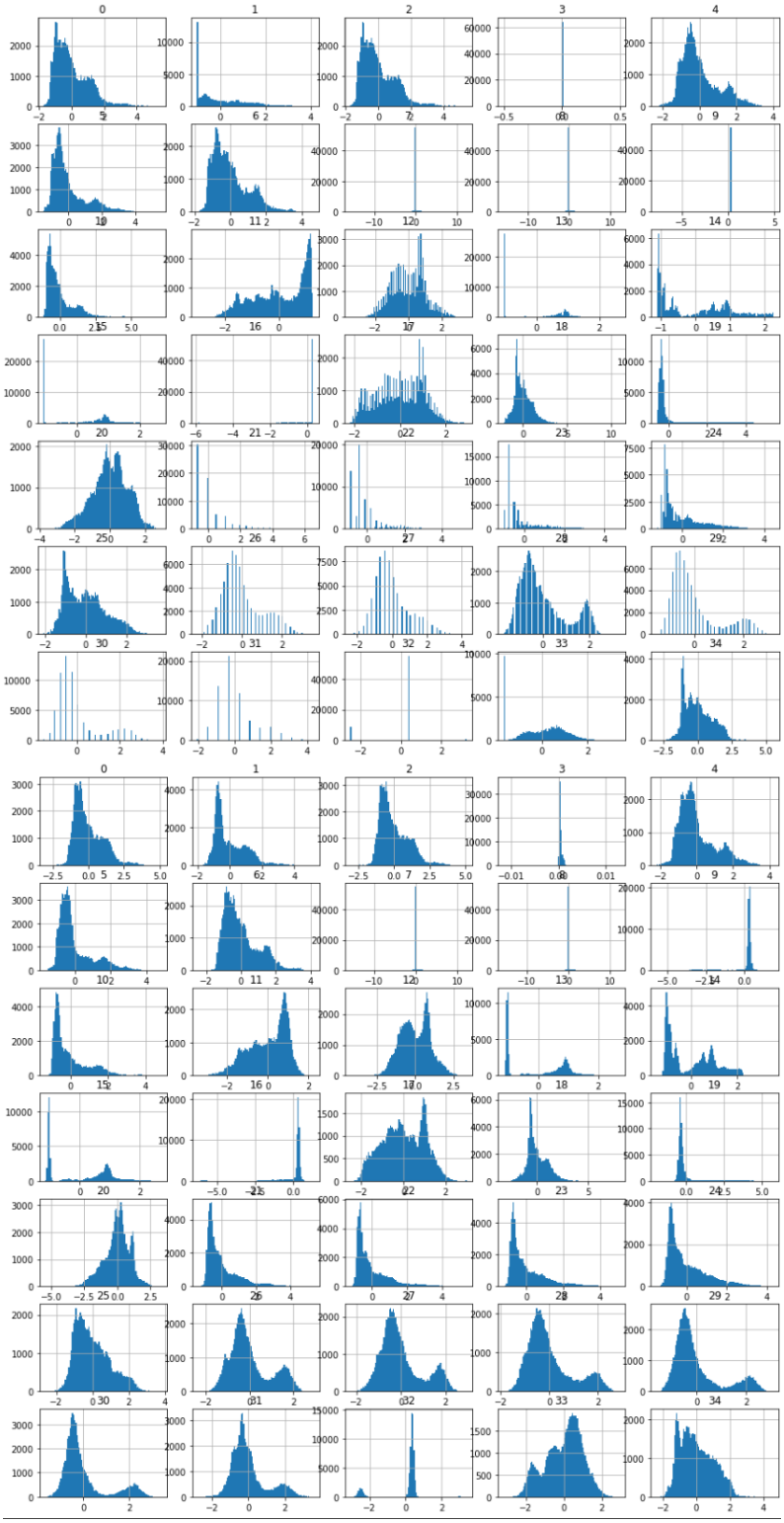


Figure 4.25: Comparison of original data distributions with generated data distributions

Chapter 5

Conclusions and future improvements

5.1 Conclusions

Before starting a conclusive discussion that closes the circle on the results obtained in the previous chapter, I prefer to take a step back and start from the initial idea behind this project and how the results obtained in the experimental phases have directed the subsequent steps presented. The initial goal was to obtain a calibrated classifier that could report a degree of confidence in the prediction. Therefore, in cases where the prediction was negatively influenced by the presence of anomalies, this could be seen from the confidence score. In order to verify the possibility of achieving such a goal, the phases defined at the beginning were:

- Apply variable noise to a time series dataset.
- Extract features from the time series and obtain two datasets, one of clean series and one of noisy series.
- Perform data mixing at different percentages to obtain a final test dataset with both clean and noisy data.
- Train a simple classifier on these datasets.
- Develop anomaly detection models (GMM, VAE).
- Determine a correlation between the output score of the anomaly detector and the classifier score.
- Once a correlation has been observed, use Bayes' rule that takes as input the prediction made by the classifier, i.e., $p(y|x)$, the probability density estimated by the detector $p(x)$, and the known prior $p(y)$, to obtain $p(x|y)$. This statistical data can then be used to retrain the classifier by giving less weight to the data with low p(x|y) or even removing those data that could degrade the performance of the classifier.

Determining the correlation between the output scores of the detector and classifier became a central part of the project, as the results obtained were not in line with expectations. The output score of the detector follows a particular trend or even becomes flat in some cases as the noise increases. This type of trend does not occur when the noise is applied directly to the extracted features instead of the time series.
The feature extraction part was performed using a pre-trained extractor provided by the tsfresh library, and the first conclusion was that the extracted features are robust to noise. This conclusion reinforces the Axyon team's assumption that engineered features are more robust to noise than time series, reason why financial time series in Axyon are subject to extracion. The results of the subsequent experimentation phases also support this assumption, where the analysis of time series was performed using convolutional neural networks.
Finally, I decided to test another popular anomaly detector in literature, namely the Variational Autoencoder, and the same phenomenon as in the previous phases occurs. In other words, if the noise is injected before the feature extraction phase, the detector is not able to detect the noisy data, if the noise is injected after the feature extraction phase, the detector is able to detect the noisy data.
In conclusion, I would like to make the following observation: if the initial goal was to obtain a calibrated classifier using an anomaly score, the methods I experimented with do not allow it in the case of searching for anomalies in time series subject to feature extraction.

5.2 Future improvements

The first improvement I would like to propose concerns not going through the feature extraction phase, something that was not immediately considered in this paper in order to be consistent with the approach of Axyon AI, which, as already mentioned, processes time series by making predictions on the extracted features.
Secondly, I would like to propose alternative approaches presented in the literature. Starting with ODIN, Outlier Detection in Neural Networks. Odin (Out-of-Distribution Detector for Neural networks) is an outlier detection network that was introduced in a paper published in 2017 by Liang et al. Odin is designed to detect out-of-distribution (OOD) samples, which are samples that are significantly different from the training data that a neural network was trained on. The key idea behind Odin is to use the temperature scaling technique, which involves scaling the logits (the pre-softmax outputs) of a neural network by a temperature parameter. By doing so, the probability distribution of the network's outputs becomes softer, and the differences between in-distribution (ID) and OOD samples become more pronounced. The temperature scaling technique has been previously used to improve the calibration of neural networks, but Odin uses it in a novel way to detect OOD samples. In addition to temperature scaling, Odin also uses a perturbation approach to further enhance its outlier detection capabilities. Specifically, it applies a small perturbation to the input of a neural network and computes the change in the output probabilities. OOD samples are expected to exhibit larger changes in the output probabilities than ID samples, and Odin uses this difference as a signal to detect outliers. Odin is a simple and effective method for

detecting OOD samples, it has been evaluateed on various dataset in a wide range of fields where outlier detection is important, in particular as computer vision, natural language processing, and speech recognition.
The other proposal as a possible future approach to the problem is contrastive learning. Contrastive learning is a popular technique used in machine learning to learn representations that capture similarities and differences between data points. In the context of anomaly detection on time series data, contrastive learning can be used to learn a representation of the normal behavior of the time series, and then use this representation to identify anomalies. The basic idea behind contrastive learning is to learn a representation of a data point by contrasting it with other data points in the dataset. In the case of time series data, the contrastive learning algorithm would take a window of time series data and try to learn a representation that distinguishes this window from other windows of data. The algorithm would repeat this process for different windows of data, building up a representation of the time series. Once the algorithm has learned a representation of the normal behavior of the time series, it can use this representation to identify anomalies. The algorithm would compare each window of data in the time series to the learned representation of the normal behavior. If the window of data is significantly different from the normal behavior, the algorithm would flag it as an anomaly. There are many different contrastive learning algorithms that can be used for anomaly detection on time series data, including autoencoders, siamese networks, and triplet networks. These algorithms differ in their architecture and training objectives, but they all share the goal of learning a representation that captures the similarities and differences between data points.

Bibliography

[1] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. ACM computing surveys (CSUR), 41, pp. 1–58, 2009.

[2] Douglas M Hawkins. Identification of Outliers. Springer, 1980.

[3] Cameron Wolfe. Confidence Calibration for Deep Networks: Why and How? . 2018.

[4] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. International Conference on Machine Learning, 2017.

[5] Sunil Thulasidasan, Shafin Rahman, Zhongzheng Cheng, Yoshua Bengio, and Kevin Murphy. On mixup training: Improved calibration and predictive uncertainty for deep neural networks. Advances in Neural Information Processing Systems, 2019.

[6] Muhammad Naeem and Waqar Hussain. Measuring Calibration in Deep Learning. Neurocomputing, 474, pp. 472–482, 2022.

[7] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Enhancing the reliability of out-of-distribution image detection in neural networks. International Conference on Learning Representations, 2018.

[8] Amirata Ghorbani, Yifan Zhou, Yifan Ye, Bo Li, and Lina Jiang. MixMDN: Out-of-Distribution Detection Using Gaussian Mixture Models. Neural Networks, 2022.

[9] Eric Deng. Gaussian Mixture Models | Clustering Algorithm Python. 2021.

[10] Kumar Neeraj. Gaussian Mixture Models: What are they \& when to use? - Data Analytics . 2022.

[11] Joseph Rocca. Understanding Variational Autoencoders (VAEs). Towards Data Science, 2021.

[12] SeongUk Jang, Jaewoo Kim, and Changick Kim. Deep Clustering with Variational Autoencoder . Advances in Neural Information Processing Systems, 2019.

[13] Pranav Rajpurkar, Awni Y Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y Ng. Cardiologist-level arrhythmia detection with convolutional neural networks. arXiv preprint arXiv:1707.01836, 2017.

[14] Mohammad Javad Shamsollahi, Ravi Srivastava, and Zhangyang Wang. Enhancing The Reliability of Out-ofdistribution Image Detection in Neural Networks. arXiv preprint arXiv:2007.09615, 2020.

[15] Xi Chen, Cheng Xie, Hongyu Ma, Yulan Liao, Dong Wang, Yong Xia, and Xiaoming Liu. ECG Heartbeat Classification: A Deep Transferable Representation. 2018.

[16] Physionet. MIT-BIH Arrhythmia Database. 1999.

[17] Maximilian Christ, Nils Braun, Julius Neuffer, and Andreas W. Kempa-Liehr. Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package). Neurocomputing, 2018.
[18] Things Solver. Time series Anomaly Detection using a Variational Autoencoder (VAE). , 2020.
[19] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. The Journal of Machine Learning Research, 2013.
[20] Muhammad Munir, Saqib Ali Siddiqui, Andreas Dengel, and Sheraz Ahmed. DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series. IEEE Access, 2020.
[21] Rui Wang, Chongwei Liu, Xudong Mou, Kai Gao, Xiaohui Guo, Pin Liu, Tianyu Wo, and Xudong Liu. Deep Contrastive One-Class Time Series Anomaly Detection. arXiv, 2022.