UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale in Sistemi Informativi

Predictive Maintenance via Anomaly Detection: an approach based on conditional autoencoders

Anno Accademico 2019/2020

Relatore
Ch.mo prof. Vincenzo Moscato
Ing. Giancarlo Sperlì
Ing. Antonio Galli

Candidato
Emanuele Di Fiore
matr. M63000921

Abstract

Predictive Maintenance is one of the most important field in the Industry 4.0 era. With the use of Internet of Things (IoT) tools and Information and Communication Technologies (ICT), an huge amount of data related to industrial machinery operating conditions can be collected, enabling and encouraging the use of data-driven approaches for predictive maintenance, especially those based on machine learning and deep learning techniques. With this knowledge, maintenance activities can be planned in optimal way, reducing or even avoiding downtime and saving costs due to unnecessary maintenance procedures. Anomaly detection is a set of techniques used to understand if conditions in which an industrial machine is working are anomalous or not, only using data collected with a various set of sensors. In last years, many researches started to use deep learning approaches for anomaly detection purposes, encouraged by the great results of these techniques in Computer Vision (CV) and Natural Language Processing (NLP). For example, one of the most used deep learning models is Autoencoder. In fact, even if it is mainly used for dimensionality re-

duction, autoencoder can be easily adapted to anomaly detection tasks especially because it allows to resolve its main problem: the absence of sufficient anomalous data to build a classifier. An autoencoder can be trained in unsupervised way, with the absence of anomalous data, using only measurements recorded by sensors from normal state working machines.

Usually, data used for anomaly detection are collected using IoT sensors and range from time-series temperature measurements to equipment vibrations, like bearings, collected by accelerometers. A recent trend is to train autoencoders for anomaly detection using audio clips, collected by microphones, which contains sounds coming from machinery. In literature, this task is called Anomalous Sound Detection (ASD). Therefore, in this text an unsupervised anomalous sound detection framework based on conditional autoencoders is presented. The main idea is to train the models on normal sounds recorded from different machines of the same type (for example industrial pumps), using their identifiers for autoencoder latent representation conditioning. In this way, only one anomaly detector per type can be implemented for the detection of suspicious working states (for example a detector that can be used for anomaly detection related to four pumps).

To validate the framework, the MIMII part of DCASE 2020 Task 2 Challenge dataset is used as a benchmark, because it natively designed for unsupervised anomalous sound detection task. In particular, on

the basis of how autoencoder is built, two instances of the proposed framework are implemented: ID Conditioned Convolutional Autoencoder and ID Conditioned LSTM Autoencoder. Results are compared with the baseline model results, provided by the authors of the challenge, and with those found in papers published by the participants. AUC and partial AUC (pAUC) are used as comparison metrics.

In conclusion, an online version of the framework is proposed to enable real-time anomaly detection.

Contents

List of Figures

List of Tables

Chapter 1

Anomaly Detection in Industry 4.0

The first part of this chapter gives some essential definitions about the paradigm of Industry 4.0 and what are its advantages, especially in the context of maintenance. After this brief introduction, an overview of the different approaches to maintenance is presented. Next, the focus is moved to anomaly detection, of which definitions and challenges that must be faced are provided. The chapter ends with an overview of some approaches used in literature to solve anomaly detection tasks, with a spotlight on autoencoder models, providing also the theoretical notions needed to understand the use cases presented in the next chapters.

1.1 Introduction

The term "Industry 4.0" was used for the first time at the Hannover Fair of Industrial Technologies, in 2011, and refers to the Fourth Industrial Revolution. According to one of the most common definition in literature, Industry 4.0 is a paradigm, a new and innovative way of implementing the industrial processes for the production of a good or a service, thanks to the introduction of the most modern information and communication technologies (ICT), like Cyber Physical Systems (CPS), the Internet of Things (IoT) and Big Data analytics methods. The introduction of ICT into core industrial processes allows to take a step forward to the computerization or automation, already introduced with the third industrial revolution, especially in the manufacturing industry: a simple factory becomes smart with integration of different physical and digital systems, from the production sectors to marketing or logistics ones, through devices, called smart sensors. Smart sensors allow the creation of a machine-to-machine interaction without the interventions of human operators [21], enabling also the collection of a large amount of data like measurements of temperature, humidity, vibration speed or sound of an equipment for monitoring automations and process configurations improvements. This large amount of data can also be useful to assess the current health status of machines and to plan maintenance activities, which have the main goal of reducing the unexpected downtime and expensive costs due to failures occurrences. Moreover, this new paradigm helps the improvement of new business models and allows to satisfy the emerging demand of products customization through an intelligent process control and management.

1.2 Maintenance approaches

Maintenance is the combination of all technical, administrative and managerial necessary actions during the life cycle of an item (in our context a machine) intended to restore it to a state in which it can perform better what it is designed for [26]. It is clear that maintenance activities are not only the physical operations that are needed to repair a particular machine, but also all activities for the planning and the scheduling of such operations, together with cost analysis. For these reasons, maintenance plays an important role in ensuring the success of a manufacturing company due to its impact on productivity, quality of products and companies economic balances.
In literature, several maintenance approaches can be found [4]. Three strategies are listed below, also summarized by the Figure 1.1:

- *Run-to-Failure (R2F)* or *Corrective maintenance* consists in repairing an equipment when it stops working. This is the simplest strategy, but also the less convenient one because it is necessary to stop the production in order to repair or replace the components that create problems.

- *Preventive Maintenance (PvM)* or *Time-based Maintenance* is more effective than R2F but it increases the operating costs because some times these corrective actions are scheduled when unnecessary.

- *Predictive Maintenance (PdM)* uses some prediction tools to establish when maintenance is necessary. It allows a continuous monitoring of machines status and an early detection of failures, avoiding unnecessary actions.

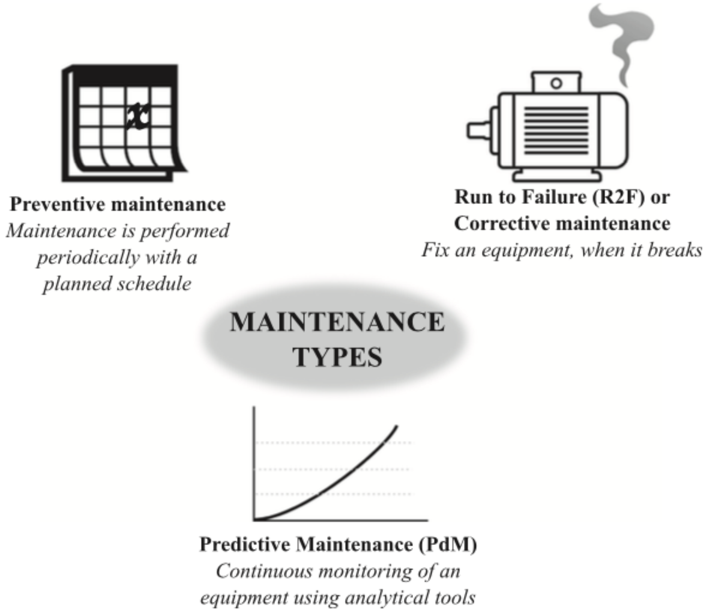

Preventive maintenance
Maintenance is performed periodically with a planned schedule
Run to Failure (R2F) or Corrective maintenance
Fix an equipment, when it breaks
MAINTENANCE TYPES
Predictive Maintenance (PdM)
Continuous monitoring of an equipment using analytical tools

Figure 1.1: Overview of different maintenance strategies [4]

Therefore, an optimal maintenance strategy should improve the equipment conditions for a better quality of final products, reduces equipment failure rates to minimize downtime in production, maximizes equipment lifetime and minimizes the total costs. According to

this, the PdM is the most promising strategy and the one that is under the researcher's spotlight in the last few years.
Citing the study realized by Thyago P. Carvalho et al. [4] and Weiting Zhang et al. [27], the PdM methods are mainly divided into three categories:

- *Model-Based Predictive Maintenance* consists in the development of complex mathematical models which replicate the behaviour of an equipment and its degradation process.
- *Knowledge-Based Predictive Maintenance* consists in the application of threshold-based rules, built on some measurements taken from machines, and that generate alerts in case in which some of them cross the thresholds.
- *Data-Based or Data-Driven Predictive Maintenance* makes the use of progresses reached in the fields of advanced analytic and artificial intelligence (AI) techniques to build machine learning (ML) or deep learning (DL) models that have the capability to predict when next failures will occur.

It is good to clarify that behind the word "predictive" several meanings are hidden. In data-driven techniques the main goal is to train ML or DL models on historical data, which contains information about the degradation of a component or about normal or anomalous behaviour of an equipment, and then use them to predict its remaining useful life

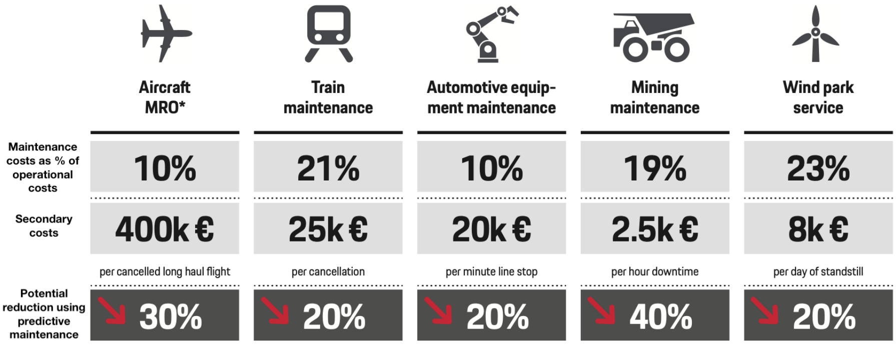
(RUL) or to detect some anomalies and generate alerts. Obviously this category of PdM is boosted by the paradigm of Industry 4.0 thanks to the large amount of historical or real-time time-series data collected by sensors in smart factories, allowing ever better performances.

The main drawback of using Industry 4.0 paradigm, especially to enable predictive maintenance, is the cost of digitalization, but fortunately in last years European Governments has started to support this transition defining new and accurate plans of investment in research and development. For example, the Italian government, with the "New Transition Plan 4.0", published in 2020 and valid for the next three years, is going to invest around 24 billion euros to promote the digital transition and to support investments by private companies [8].

In conclusion, regarding tangible results of the transition, is worth analyzing the Figure 1.2, taken from the study made by Porsche Consulting [15], which results show a significant reduction in maintenance costs through implementation of predictive systems.

1.3 Anomaly Detection in PdM

Anomaly detection (or outlier detection) is a set of techniques for the identification of anomaly patterns in data that deviates from normal behaviour. For example, a machine, like a pump or a steam turbine, after a period of normal behaviour starts deteriorating due to regular



	Aircraft MRO*	Train maintenance	Automotive equipment maintenance	Mining maintenance	Wind park service
Maintenance costs as % of operational costs	10%	21%	10%	19%	23%
Secondary costs	400k €	25k €	20k €	2.5k €	8k €
	per cancelled long haul flight	per cancellation	per minute line stop	per hour downtime	per day of standstill
Potential reduction using predictive maintenance	↘ 30%	↘ 20%	↘ 20%	↘ 40%	↘ 20%

Figure 1.2: Potential overview for maintenance cost reduction across five industries [15]

use, generating some *anomalies* and entering in a status that can be identified as anomalous. This state should not be considered as a total failure state (in such case the machine should be turned off) but as a warning state, indicating that some maintenance procedures are needed [13]. Anomaly detection is so used to minimize the cost of failures, discovering anomalous behavior of mechanical devices in the production line in order to predict potential problems as early as possible. In this way, the production plans can be adjusted accordingly to avoid failures, and the failures already happened can be contained in their early stage to avoid cascading effects.

As for the PdM in general, in last years, in the context of anomaly detection, different ML and DL techniques have been developed. Most of them are based on time-series, which represents historical or real-time measurements of different parameters recorded in sequence from machines by smart sensors. In this type of data, three different kinds of anomaly can be identified [6]:

- *Point anomaly*: the time-series returns in normal state in very short time period, so the anomaly is represented by only few observations;

- *Contextual anomalies*: observation or sequences that deviates from expected patterns, but if taken in isolation they not exceed the range of expected values for that signal;

- *Collective or Pattern Anomalies*: observations that are labeled as anomalous only if taken together.

In the Figure 1.3 can be found a graphical view of the just mentioned anomaly types.

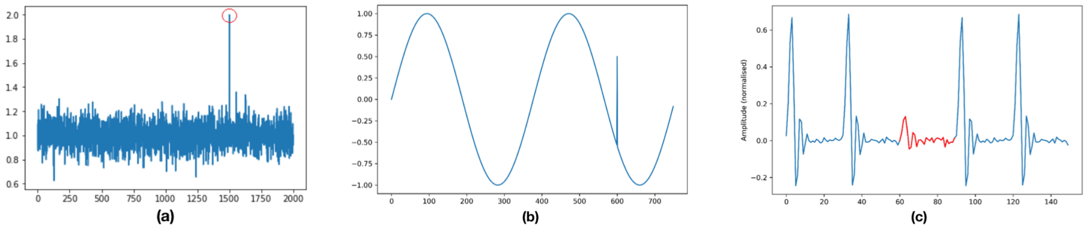


Figure 1.3: Point anomaly (a), contextual anomaly (b) and collective anomaly (c) [6].

1.4 Challenges of Anomaly Detection

Being a field of predictive maintenance, also in anomaly detection there are some important elements that must be taken into consideration, because they influence the performances of trained models.

1.4.1 Contextual information

The presence of a variety of sensors, distributed around the environment of the monitored system, gives the opportunity to include contextual information into the collected data, allowing the improvement of the anomaly detection process performances. In particular, two types of contexts must be handled: spatial context and external context. When multiple sensors are deployed to monitor a system that moves in different environment conditions, such as a train, the contextual information can effect a lot the performances of an anomaly detector. For example some behaviours that are normal when a train is running on a flat ground can be anomalous when it is climbing an incline. This problem could be resolved using an accelerometer that measures the angle with the ground and incorporate its observations during the training. Moreover, external context can be likewise effective when monitoring the internal temperature of an equipment, because in this case the knowledge of external temperature can effect the results of the detection in better. Obviously, taking in consideration the context, some drawbacks are present, for example the necessity to build a more complex and expensive monitoring system and a more difficulty in model training.

1.4.2 Data dimensionality and noise

Other important elements that must be taken in consideration into the development of an anomaly detector are the data dimensionality and measurements noise. Regarding data dimensionality, there are two different cases: univariate or multivariate data. Univariate data consists of observations recorded by a single sensor, while multivariate data consists of a sequence of observations recorded by multiple sensors. In the first case, machine learning models must be trained to discover the historical relationships that could be hidden in the sequence of values recorded by one sensor. In the second case, observations recorded by more than one sensor are linked together by the timestamp. This can be regarded as an advantage in some situations in which patterns are hidden between the relationships among the various physical measures monitored by sensors. Regarding the noise, in an IoT environment where a large number of low cost, resource constrained sensors are deployed, the data quality is often affected by significant disturbs, inconsistencies and missing or duplicated data. Where the sensors are powered by battery, these challenges are often amplified as the available charge decreases. These problems can be resolved aggregating data from multiple similar sensors into a single observation to reduce the environmental noise.

1.4.3 Stationarity

Stationarity is also another important element that needs to be treated: a stationary time-series is one where the mean, variance and autocorrelation does not vary with time. Unfortunately, in real world non-stationary time series are very frequent and this make more complex the training of AI models because statistical data stream distribution may vary over time and so what is normal over a period of time could be anomalous in another (concept of seasonality).

1.4.4 Lack of anomalous data

One last challenge, maybe the most important that must be faced in anomaly detection, is the prediction of the "unknown". The word "unknown" refers to undesirable (or anomalous) events of which available data are not enough. The consequence is that ML and DL models can not be trained using a classification approach, because only normal state monitored data of an equipment are available to build an anomaly detector [25]. In fact, for example, while it is possible to build a toy industrial machine in order to collect data of its normal and anomalous working states, on the other hand, in some situations, it is unthinkable, dangerous and expensive try to generate anomalous behaviors of a machinery used in real world, like an aircraft engine, a train engine or an industrial press, with the only goal of collecting data. Moreover, even if anomalous data are available, they not surely represent all the anomalous situations in which the machine under investigation could be, because the space of the anomalies is very irregular and it is practically impossible to have a complete vision of them in order to trains models with a classification approach.

1.5 Unsupervised Anomaly Detection

Because is quite simple to collect large dataset associated to a particular machine's working state, in such situations, ML and DL models must be trained in a unsupervised way, where the word unsupervised means that the models try to detect an events that have no examples in the data history used for training [24]. At this point, a question that arises is: how a trained model could detect never seen anomalous measurements and generate alerts? A possible approach is showed in Figure 1.4. The figure shows that models could be trained to reconstruct the received input, that refers to a normal working state, and if the output is enough different from the input an alert will be triggered.

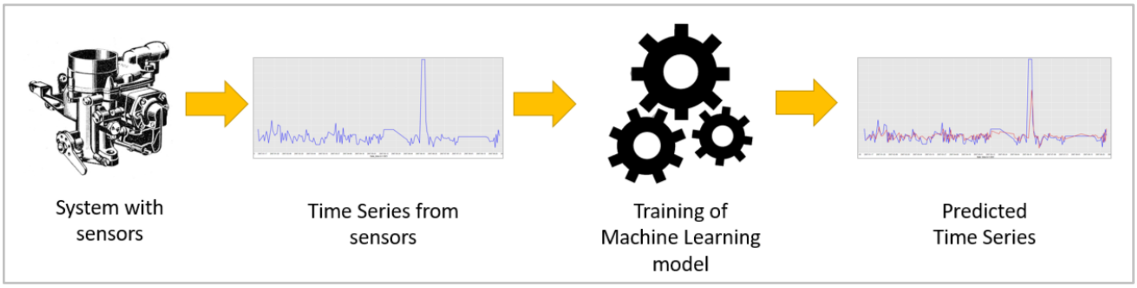


Figure 1.4: Anomaly detection high level workflow [25]

Anomalies in sensor data can be defined as previously unseen patterns and the algorithm for anomaly detection should be able to detect known anomalies as well as generalize to new and unknown ones. In general, a common approach to achieve this detection is a sliding-window (Figure 1.5). Since models are trained on normal data which is a fixed-length sequence of preceding steps, it can achieve anomaly detection by comparing the predicted value at each timestep with the actual sequence, allowing also to perform a real-time detection and eventually alerts generation [11] and, moreover, using the temporal relationships present in sequences. Several approaches have been developed for unsupervised anomaly detection. R. Silipo et al [24] describes two techniques: the first, named Control Chart, and the second based on Auto-Regressive (AR) models. In the first case, the boundaries for an anomaly-free functioning equipment are defined. This boundaries are usually centered on the average signal values recorded by sensors and bounded by twice the standard deviation in both directions. If the signal is wandering off this anomaly free area, an alarm should occur. The second approach is conceptually similar to the Control Chart and uses this anomaly-free time window to train an AR model.

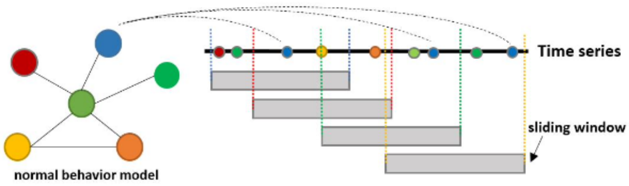


Figure 1.5: Online detection procedure with sliding window [11]

The boundaries for anomaly-free functioning are defined here on the prediction errors on the training set. To clarify, the model receives in input a time-series made by sensor observations and the next time-step value is then predicted by the model. Successively, according a threshold established using the training set, if the predicted value is enough different from the actual value, an alarm is fired off requiring further checkups. This second technique is also cited in the survey [6]. Another important and suitable machine learning method for this task is **autoencoder** and for this reason it will be at the center of this discussion. An autoencoder (Figure 1.6) is a neural network mainly designed to encode the input into a compressed and meaningful representation and then decode it back, such that the reconstructed input is as much as similar to the original one [2].

Autoencoder is so an unsupervised technique because any label or class is needed for training phase, so it results a perfect approach for anomaly detection. In details, the encoder part is made of a series of layers with a decreasing number of nodes and ultimately reduces data

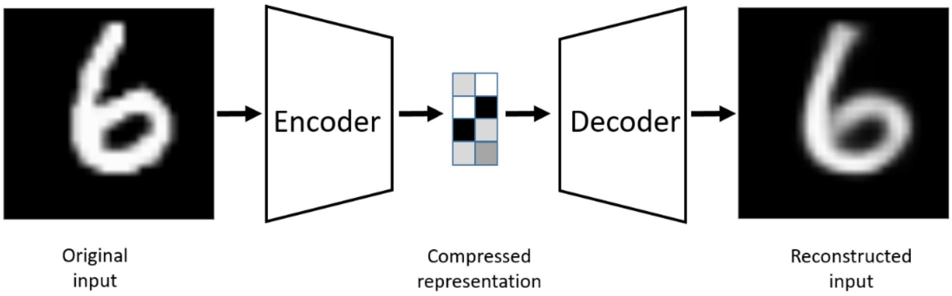


Figure 1.6: An autoencoder example [2].

into a latent vector, which represents a reduced (encoded) version of the input and which contains only valuable or essential information of it. The decoder, as the encoder, is made of different layers with an increasing number of nodes. It is trained to receive in input the latent vector and to generate an output as much as possible similar to the input. The process is described in mathematical way as follow:

$$argmin_{D,E}||X - D(E(X))||$$

Where X is the input data, E is an encoder network, and D is a decoder network. Using this formula is evident that the performances of an autoencoder are evaluated using the reconstruction error: the difference between the input and the output in terms of mean absolute error (MAE) or mean squared error (MSE).

Summarizing, in the context of anomaly detection in predictive maintenance, the autoencoder can be trained to reconstruct the normal observations collected from a working machine in normal working state. Next, in detection phase, its reconstruction error can be used as an anomaly score: if it is under a threshold the observation in input can be classified as normal, otherwise as anomalous. As mentioned before for AR models, the threshold is found using the reconstruction error of training set samples or using different techniques.

The next chapter presents different autoencoder approaches in anomaly detection, with a particular focus on anomalous sound detection tasks.

Chapter 2

Related Works

This chapter presents, in the first section, an overview of the different datasets available online for anomaly detection tasks. Next, different autoencoder-based architectures found in literature for anomaly detection are presented.

2.1 Available datasets

One of the main reasons for the success of ML or DL algorithms is the availability of more and more public standard datasets, regarded as common benchmarks and that allow a fair comparison between different proposals present in literature. In PdM and in anomaly detection tasks, this is not totally true, because often machinery data are confidential (so not publicly available) and sometimes studies are conducted by companies internal divisions. In response to the need of public dataset arranged for anomaly detection tasks, in last years some datasets have been built and then made available online (Table 2.1).

Dataset Name	Summary
CWRU	Ball bearing test data for normal and faulty bearings. Motor bearings were seeded with faults using electro-discharge machining (EDM) and vibration data was collected using accelerometers, which were attached to the housing with magnetic bases.
MFPT	Provides time-series data from nominal, outer race fault at various loads, inner race fault at various loads and three real-world faults of bearings
NAB	Contains cross-domain data collections, like the average CPU usage in AWS cluster or the internal temperature data of an industrial machine.
Yahoo S5	Consists of four data classes, each of which contains either a set of synthetic or real web traffic metrics tagged with anomalies
DCASE2020	The dataset consists in sounds collected from different industrial machines. It is composed by MIMII Dataset and ToyADMOS.

Table 2.1: Anomaly detection online available datasets.

To clarify, these datasets contain some anomalous observations, often generated manually by operators, but, as mentioned in the first chapter, they not represents all the possible anomalous states. Anomalous observations are mainly used to build test set and consequently evaluate the performance of the models, trained in unsupervised way.

2.2 Anomaly detection with autoencoders

In this section, some autoencoder-based architectures found in literature are presented. Antonio L. Alfeo et al. [1] show an autoencoder-based approach using power consumption of industrial laundry assets and bearing vibrations data. In the first case, time-series are realistically created using official information provided by assets manufacturers (e.g. nominal energy consumption) and by researchers' industrial partner. In the second case, time-series are extracted from MFPT dataset presented in Table 2.1. After the creation of these ad-hoc datasets, the proposed approach employs an autoencoder to score the anomaly degree of each input instance, presented as rescaled time-series features. The reconstruction errors, calculated downstream of the decoder, are further processed by a discriminator, which rescales them between 0 and 1 by using a sigmoidal function (Figure 2.1). For both the dataset, the inputs of the architecture are samples characterized by statistical features extracted from time-series available in the dataset which capture their trends over time (like 90th, 75th, 50th and

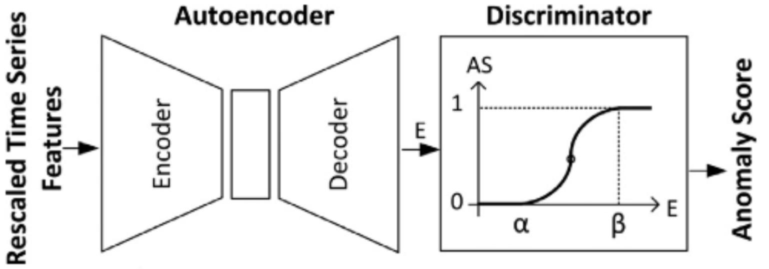
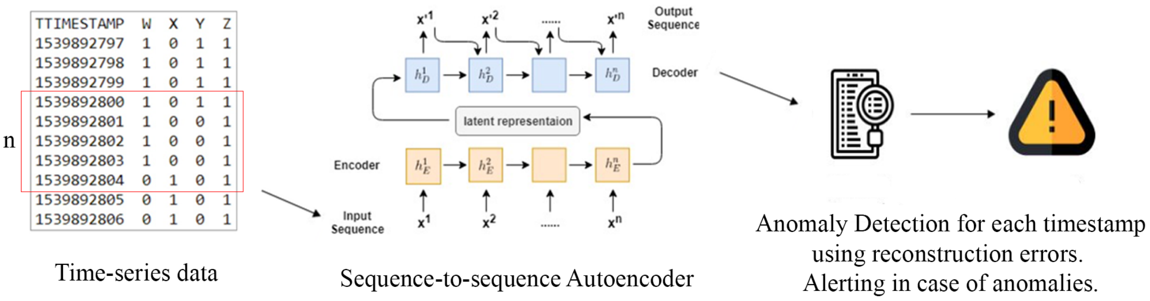


Figure 2.1: Anomaly detector architecture proposed by [1].

25th percentile of the time series or the mean absolute deviation).

Ruei-Jie Hsieh et al [11] developed an unsupervised deep Recurrent Neural Network (RNN) detection model based on the Long Short Term Memory (LSTM) autoencoder. The recurrent layers of the autoencoder capture temporal dependencies in multivariate sensor data provided by a manufacturing company. The mentioned company has a production line in which there are three chambers (A,B,C) disposed in series and the product moves from a chamber to another. Each chamber has 4 sensors (W,X,Y,Z) for the detection of the products passage. When sensors outputs signal 1, it indicates the product is already in a chamber, otherwise it outputs 0. Obviously, the anomalies in data are not detectable in the single values generate by sensors, but only in the time sequence of them. This is the reason why the authors have chosen LSTM cells to build encoder and decoder layers. The detection of an anomalous behaviour in a chamber prevents that the products pass in the whole line when there is something wrong in the production process (early detection). Autoencoder models are trained to reconstruct input sequences, timestamp by timestamp, using a sliding-window method (sequence-to-sequence autoencoder), already analyzed in the first chapter. The input vector in each timestamp is composed by the readings of the four sensors in a chamber. Using a threshold ϵ, defined through the reconstruction error of training data, each timestamp of each sequence is labeled as normal or anomalous on the basis of its reconstruction error. In conclusion, because each timestamp is labeled as many time as the sliding window length, a majority voting mechanism is applied to make the final decisions and eventually an alert is generated (Figure 2.2).



TTIMESTAMP	W	X	Y	Z
1539892797	1	0	1	1
1539892798	1	0	1	1
1539892799	1	0	1	1
1539892800	1	0	1	1
1539892801	1	0	0	1
1539892802	1	0	0	1
1539892803	1	0	0	1
1539892804	0	1	0	1
1539892805	0	1	0	1
1539892806	0	1	0	1

Figure 2.2: Architecture pipeline proposed by [11].

Another strategy to resolve the anomaly detection task is to combine deep learning and ensemble learning. Shao Haidong et al. [23] propose a novel method called ensemble deep autoencoders (EDAEs) for the intelligent fault diagnosis of bearings, using CWRU dataset. The ensemble of multiple deep autoencoders is a good choice to overcome the low generalization ability of individual deep autoencoders. Diversification is obtained by designing deep autoencoders with different kind of activation functions, considering that neural networks differing in activation functions usually show different characteristics and complementary learning behaviours. Regarding combination strategy, the authors define a new one. Firstly, for each deep autoencoder the classification accuracy is calculated and then compared to a pre-defined threshold. Next, only autoencoders with accuracy greater than the threshold are kept, then different weights are calculated and assigned to each model using its accuracy. Finally, the achitecture reports the combined diagnosis result of each sample based on the weight scores. In order to maintain the stability of the combined diagnosis results, repeated trials are carried out. The results, in terms of precision, recall and F-Measure, demonstrate the validity of this architecture.

Chen et al. [5] introduce an unsupervised anomaly detection system for industrial robots, based on a sliding-window convolutional variational[1] autoencoder (SWCVAE), which realizes real-time anomaly detection by coping with multivariate time series data. Their architecture can be can be trained in unsupervised manner and then used to detect spatial and temporal anomalies in data. To validate the approach, KUKA KR6R 900SIXX is chosen by the authors, which is an industrial robot with six revolute joints that repeatedly performs a pick-and-place task using vision guidance system. The authors record the joint angles and joint currents of the six joints of the robot at 33.3 Hz from the robot controller, so without using external sensor devices but only the data provided by the robot itself. For testing purpose, in order to simulate faults, collisions have been induced by manually hitting the robot.

[1] Differently from classical autoencoders, in VAE, the latent variable z is constrained to be distributed according to a prior distribution $p_\theta(z)$, usually multivariate unit Gaussian $\mathcal{N}(0, I)$, forcing the model to learn the distribution of input data.

2.3 Anomalous Sound Detection (ASD)

The existing anomaly detection systems used in the industrial domain depend, obviously, on the properties of sensors used to monitor an industrial machine. Among those systems, most common are visual anomaly detection systems, which have some drawbacks such as illumination, occlusion by objects, being out of the field of view and so forth, which strongly affect the performance of the system, especially in terms of computation power needed to achieve high real time performances. ASD systems, however, are not affected by the problems just described. In fact, the use of acoustic data features also offers an advantage in terms of simplicity in data retrieval. On the other hand, compared to images, audio clips need some pre-processing steps to be converted in different formats compatible with neural network training process, like signals representing air pressure values or spectrograms, often in the Mel scale[2].

In this context, the authors of [3] compare the performances of convolutional LSTM autoencoder (Conv-LSTMAE) and sequential convolutional autoencoder (CAE) using sounds retrieved from Youtube videos recorded during industrial manufacturing processes. Because it is hard to capture abnormal patterns due to their rarity in real life and because the creation of anomaly events and recording respective sounds

[2] In Mel-Scale the entire frequency spectrum is separated into x evenly spaced frequencies (bins), where 'evenly spaced' means that the distance on the frequency dimension approximates the human auditory system's response more closely than the linearly-spaced frequency bands used normally in spectrograms.

is expensive, the authors generate artificially anomalous samples for testing adding anomaly events like explosion, fire or glass breaking to downloaded clips. Although the proposed two autoencoders are different in terms of layers, the pipeline of the system is fixed: there are sequences of five 128x128 spectrograms (extracted from audio dataset) that are the inputs of a sequential autoencoder, which generate output sequences that must be as much similar as possible to the input ones. Also this time, the decision about the generation of an alert due to the possible presence of anomalies, is made on the comparison between the reconstruction error of each spectrogram and a pre-defined threshold (Figure 2.3).

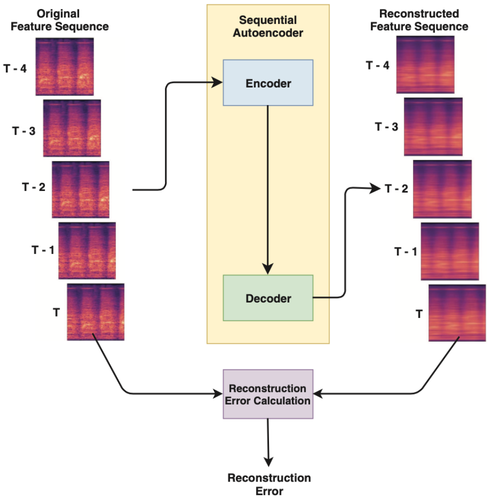


Figure 2.3: The reconstruction and reconstruction error calculation stages [3]

Another paper in which the reconstruction error of a pre-trained autoencoder is used for anomaly detection using sound data, is the one proposed by Dong Yul Oh et al. [18]. They propose a convolutional

]

autoencoder which receives in input frames of size 1024x32, obtained after a segmentation of mel spectrograms extracted from an audio dataset. The particularity of this work is in how the authors validate their approach. They use a real Surface-Mount Device (SMD) assembly machine, which performs a dozen operations in a second. Since the microphones are disposed really close to the machine, additional processing for the ambient noise is not required. The entire machine production in their experimental data consists of two lines (B and C), which differ in terms of which semiconductors are being assembled. C-Line is used as a normal category and all others are assumed to be abnormal (one vs. all). They manually defined three abnormal categories: changing assembly part (from C-Line to B-line), adding noise artificially (to represent broken internal components) and removing grease in the same production C-line. The details of the convolutional architecture are shown in the Figure 2.4.

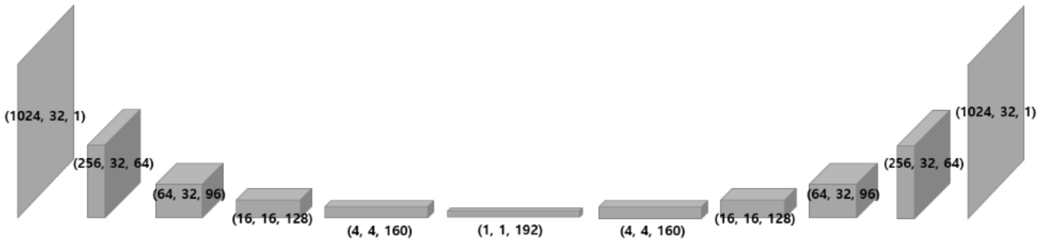


Figure 2.4: Overall structure of convolutional autoencoder proposed by the authors of [18].

The case study of the SMD machine was brought forward in [19], in which a model, called FARED is presented. FARED stands for Fast Adaptive RNN Encoder–Decoder and it is a new recurrent approach used for anomaly detection. Here, autoencoder is built using LSTM for each RNN cell.

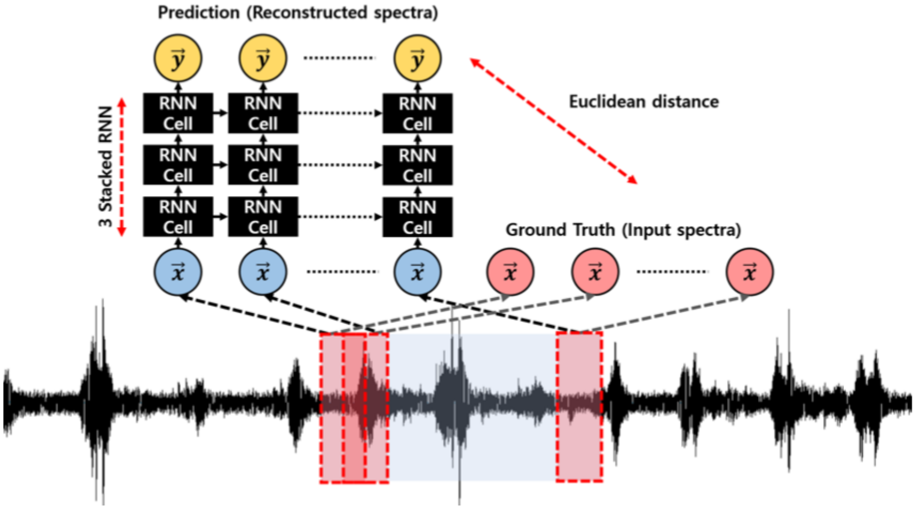


Figure 2.5: Structure of Fast Adaptive Recurrent Neural Network (RNN) Encoder–Decoder (FARED) [19].

The Figure 2.5 shows that the input is constructed by sequential spectrum (red box) and that each spectrum has 50% of overlapping in the time domain. The output is a sequential reconstructed spectrum from input sequences. The Euclidean distance between prediction and ground truth is used for training and anomaly detection. The main objective pursued with the introduction of this recurrent architecture is the training time reduction, since this type of machine is used in different small productions, dedicated to different target products. In fact, the production features often changes and models must be trained easily and as soon as possible. Both the autoencoder tested on SMD machine uses the threshold mechanism to eventually generate alerts. Very interesting is the study conducted by Maarten Meire and Peter Karsmakers [17]. They propose two approaches: one based on One-Class Support Vector Machine (OC-SVM) and one based on autoencoders trained in unsupervised way. The novelty brought in this paper does not reside so much in how the models are trained or built, but in their efficiency, in terms of accuracy but also in computational power. In fact, the main focus is often to increase the difference between the reconstruction errors of normal and anomalous data, without considering training performances and speed, but if it is necessary to bring the trained system to the edge (on sensors themselves or close to them) computational power is a factor to be monitored carefully. This study is conducted comparing different implementations of convolutional neural network autoencoders and OC-SVM approach, using acoustic data collected from healthy and faulty bearings generated with Siemens Industry Software, pre-processed to generate spectrograms in mel scale, similarly as seen in previously analyzed works. This comparison, in terms of models accuracy, training times and number of algorithms parameters, is done twice, when the microphone is close to the faulty bearing and when it is far from it. Conclusions drawn by the authors highlight that the best compromise between accuracy and performance on hardware for real-time detection is represented by the 2D-CNN autoencoder.

When an ASD system overlooks an anomaly, an update of the system is necessary to never overlook the observed type of anomalies twice.

There are two ways to face this problem: re-training the whole system using all training data, or cascading a new specific detector for the overlooked anomaly. The first approach is the most effective solution but an huge computational cost and an amount of anomalous training data are required to re-train the system. In [16] the second one is tried, with a few-shot learning method for ADS. The authors propose and architecture called SNIPER (few-Shot learNIng with ensured true-PositivE-Rate), whose goal is to maximize the true positive rate (TPR) on the observed anomaly, but the problem is the absence of an huge amount of data related to it (Figure 2.6). The solution found by the authors is the following: they combine the anomaly score produced by an autoencoder trained in unsupervised manner and a new ad-hoc similarity score, produced by a specific anomaly detector $\mathcal{S}$ on the basis of a comparison between input samples and memorized anomalous sounds (overlooked in the past). The authors build $\mathcal{S}$ using an approach based on VAE, to overcome the problem of the absence of sufficient overlooked anomalous data.

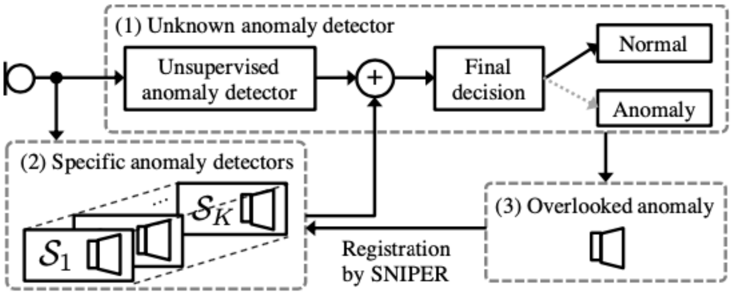


Figure 2.6: High level view of SNIPER. Overlooked anomalous sound is registered to specific anomaly detector with proposed method. [16]

2.3.1 DCASE Challenge: an ASD task

Detection and Classification of Acoustic Scenes and Events, is a community that organizes challenges every year and usually their task 2 regards predictive maintenance with audio clips as dataset. The dataset presented in the Table 2.1 is provided by DCASE 2020 Challenge, whose title is "*unsupervised detection of anomalous sounds for machine condition monitoring*". Because of only normal working machines sound samples are provided as training data (in form of clips, lasting 10 or 11 seconds), the task is already arranged to be resolved in unsupervised way, and, again, autoencoder is one of the best techniques. In details, the dataset that can be downloaded is composed by ToyADMOS Dataset and MIMII Dataset [7] and refers to different sounds recorded with microphones disposed around different machine types: a toy-car, a toy-conveyor, a valve, a pump, a fan and a slide rail. To make the challenge more interesting, different machines for each type are considered, identified by an ID string (for example different kinds of pump). A baseline system implementation for comparison purpose was made available by the authors of the challenge. It consists of a dense autoencoder with three layers, in both the encoder and decoder components, with 128 units, and a latent space with 8 units, all with the ReLU activation function. Following, a summary of most interesting approaches for the improvement of the baseline system is shown, while, in the next chapters, the dataset is explored more in details to support a better explaination of the experimental part of this text.

Pilastri et al. [22] propose two deep learning models, based on a dense and convolutional architectures fed with spectrograms. For the dense one, the encoder and decoder networsks consist of four fully-connected layers, followed by Batch Normalization and ReLU as the activation function. For the convolutional one, the encoder and decoder networks are composed by convolutional layers with Batch Normalization and the ReLU activation function after each convolution. In both architectures the goal is to minimize the reconstruction error between inputs and outputs. Regarding features extraction, for the dense autoencoder, audio samples are buffered in fixed-length 1 second intervals with a 50% overlap and, after the extraction of spectrograms 128x640 dimensional input matrix are obtained. In the convolutional autoencoder system, the spectrograms obtained from samples are segmented to form 128x32 frames, ready for training. As for previous described architectures, the classification of audio clips, is made comparing the reconstruction error (in terms of MSE) and a pre-defined threshold. Results show that for two machine types (slider and valve), the best results were achieved by the convolutional autoencoder, while the dense autoencoder provided the best results for the others.

Again, also in anomalous sound detection task, some LSTM-based autoencoders architecture can be found in literature. For example, in the technical report provided by Jalali et al. [12], the architecture encodes the information using LSTM layers with n units and then their outputs are passed into a bottleneck layer, which is also a LSTM layer with smaller size together with a repeat vector layer. A repeat vector layer repeats its input vector multiple times (the number of time steps chosen for the sequences). Next, a stack of sequential LSTM layers reconstruct the original input.

Another interesting approach is the one proposed by Tomoki Hayashi et al. [10]. Their paper presents a Transformer-based and Conformer-based autoencoder for ASD, performing sequence-to-sequence processing. With respect to the standard autoencoder, this kind of architectures can extract sequence-level information from whole audio inputs, using a methodology called self-attention mechanism, with whom Trasformer and Conformer neural networks are built. In fact, a critical disadvantage of classic sequence-to-sequence autoencoder is the inability of the system to retain longer sequences. The attention mechanism was created to resolve this problem of long dependencies, as it is an interface connecting the encoder and decoder providing to the decoder the information from every encoder hidden state. With this framework, the model is able to selectively focus on valuable parts of the input sequence and hence, learn the association between them.

As can be noted, all the approaches proposed above differ from the way autoencoders are built. In fact, the main idea is always to train autoencoders to reconstruct as well as possible the normal audio samples provided in input and then to detect anomalies when there is some inputs that are badly reconstructed by the architecture. In literature, however, there are also some proposals that are regardless of how the autoencoders are build. For example, the authors of [14][9] and also the same researchers of the last presented approach, introduce the concepts of ID conditioning and ID regression. The main idea of both is to use the information related to the particular device identifier from which the input sound clips are retrieved to influence the autoencoder behaviour.

Table 2.2 reports a summary of the approaches described in this chapter with a focus on application domains and their pros and cons.

Table 2.2: Related works summary

Application Domain	Approach	Ref.	Pros	Cons
Industrial Laundry Assets / bearings vibrations data (MFPT)	Dense AE	[1]	The discriminator avoids an iterative process of choosing the threshold.	Need to choose accurately the synthetics features needed to extract sensor data trends over time.
Production line chambers with presence sensors	LSTM AE	[11]	Capacity to capture temporal dependencies in multivariate sensor data.	Tuning needed to choose the sliding window length and hop-size.
Fault diagnosis of bearings (CWRU)	Ensemble Deep AE (EDAEs)	[23]	Overcome the low generalization ability of individual autoencoders.	High complexity of the architecture due to the number of autoencoders that must be trained.
Joint currents and joints angles of a pick-and-place robot	Sliding Window Conv VAE (SWC-VAE)	[5]	Capacity to detect spatial and temporal anomalies in data, also in real-time.	Tuning needed to choose the sliding window length and hop-size.
Industrial manufacturing processes sound retrieved online	Con-LSTM-AE and Seq-Conv-lAE	[3]	Capacity to detect temporal dependencies in data.	Longer training time due to the convolutional nature of the network
Audio data from Surface Mount Device assembly machine (publicly available on GitHub [a])	Conv-AE	[18]	Capacity to extract features autonomously from audio spectrograms.	Tuning needed to choose the sliding window length and hop-size to extract spectrograms from a audio.
	LSTM AE	[19]	Capacity to detect temporal dependencies and to allow fast training.	Tuning needed to choose sequence length and hop-size used for anomaly detection.
Sounds recorded from compressor, engine, compression pump, electric drill and a condensing unit of an air-conditioner (DCASE 2016 Dataset)	VAE + AE	[16]	Capacity to never overlook an observed type of anomaly twice thanks to a component which memorizes anomalies not recognized in the past, even if there are not too much samples of them.	Augmented overall architecture complexity. Necessity to preprocess audio clips related to working machine states.
Sounds recorded from industrial machines, like pumps, valves, slide rails and fans (DCASE 2020 Dataset)	Dense AE	[22]	Simplicity of the overall architectures.	Necessity to tune parameters regarding audio clips transformation process into spectrograms.
	Conv-AE	[22]	Capacity to capture temporal dependencies using audio spectrograms as inputs.	Longer training time regarding the convolutional version.
	LSTM AE	[12]	Capacity to capture temporal dependencies on audio spectrograms segments.	Tuning needed to choose the number of timesteps and the number of features.
	Transformer and Conformer based AEs	[10]	Attention resolves the inability of seq2seq architectures to retain longer sequences, focusing on valuable part of the sequence.	Architecture complexity in terms of parameters due to the presence of attention layers.
	Conditional Dense AE, Conditional Conv-AE and Conditional LSTM AE	[10],[14],[9]	Conditioning autoencoders can recognize audio samples recorded from different machines of the same type using external information, like identifiers.	Necessity to add external layers to process these identifiers, which should also be transformed in a compatible format. A new training loss must be defined to use the identifier information.

[a] https://github.com/DongYuls/SMD_Anomaly_Detection

Chapter 3

Proposed Framework

As already shown, one of the main task of predictive maintenance is anomaly detection, described as a set of techniques for the identification of some anomalous warning states in which an under observation machine could be, allowing a better maintenance activities planning. In this part, the task performed is firstly formalized and then the proposed architecture is illustrated. At the end of the chapter a list of technologies used during implementation is presented.

3.1 Problem definition

In this section the performed anomalous sound detection task is formalized. The proposed framework is built to classify as normal or anomalous audio clips placed in input. The training set is composed of K audio clips $\{x_1, x_2, ..., x_K\}$ in *.wav* format, recorded from different versions of a machine type M, with a duration of T seconds (for example K clips from four different pumps). An ID string is used to identify the different versions of M. Moreover, since only few audio clips are recorded when anomalies are present, they are incorporated into a different set, used for testing purpose. For the reasons explained in previous chapters, this task cannot be solved using classification, even though it seems to be a two-class classification problem. To clarify, only one model is trained to perform anomaly detection on different versions of M, using, in training phase, normal audio clips recorded from all of them. In this way the complexity of the predictive maintenance system is reduced, because one single model can be used for different machines, avoiding repeating the training operations, in any case expensive, for each machine.

Before describing the proposed model in detail, let's have a look to the implemented framework from an high level view (Figure 3.1). As it can be seen, the input consists in an audio clips recorded by microphones disposed around monitored machines, while the output is a label, *normal* or *anomaly*.

3.2 Model architecture

The proposed ASD system operations, shown in Figures 3.2 and 3.5, consist of two main phases: an offline phase, responsible of autoencoder models training, and an online operation phase, to conduct a real time

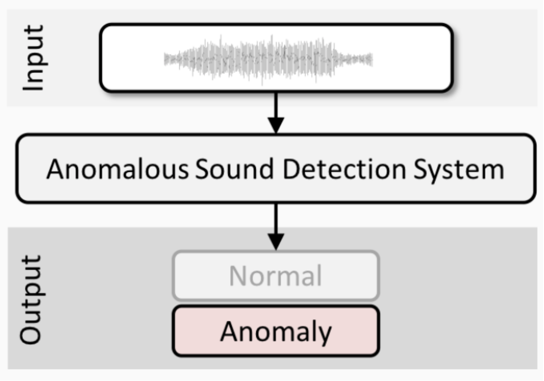


Figure 3.1: High level architecture [7]

detection.

In the next two subsections there are detailed descriptions of both phases, with a focus on each block present in the pipelines.

3.2.1 Offline Training Phase

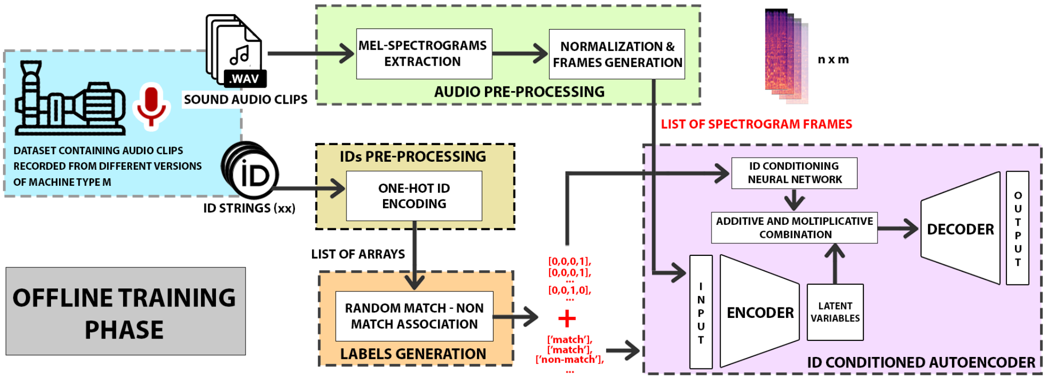

.WAV
SOUND AUDIO CLIPS
DATASET CONTAINING AUDIO CLIPS RECORDED FROM DIFFERENT VERSIONS OF MACHINE TYPE M
iD
ID STRINGS (xx)
MEL-SPECTROGRAMS EXTRACTION
NORMALIZATION & FRAMES GENERATION
AUDIO PRE-PROCESSING
n x m
LIST OF SPECTROGRAM FRAMES
IDs PRE-PROCESSING
ONE-HOT ID ENCODING
LIST OF ARRAYS
RANDOM MATCH - NON MATCH ASSOCIATION
LABELS GENERATION
[0,0,0,1],
[0,0,0,1],
...
[0,0,1,0],
...
+
['match'],
['match'],
['non-match'],
...
ID CONDITIONING NEURAL NETWORK
ADDITIVE AND MOLTIPLICATIVE COMBINATION
INPUT
ENCODER
LATENT VARIABLES
DECODER
OUTPUT
ID CONDITIONED AUTOENCODER
OFFLINE TRAINING PHASE

Figure 3.2: Overview of offline training phase of the proposed ASD system.

This phase is responsible of the training of an autoencoder. The training process must be done in offline manner with the use of a pre-collected normal audio clips which belong to machines of the same type. In addition to the dataset, analyzing the Figure 3.2, other components can be noticed: the Audio Pre-Procesing block, the IDs Pre-Processing block, the Label Generation block and the ID Conditioned Autoencoder. The first three parts prepare the dataset for training process, which it is executed in the last one.

Dataset Structure

In addition to audio clips, the dataset must provide the IDs used to identify particular versions of M, embedding them into clips filenames (with *.wav* extension) or in a separated file (obviously maintaining a one-to-one correspondence with audio files). Definitely, the data should be structured in pairs composed by audio clips plus an identifier.

Offline Audio Pre-Processing

This part of the framework is responsible of audio clips transformation, a necessary operation to make them ready for the neural network learning process, executed into the ID Conditioned Autoencoder block. There are two different components in this block:

- *Mel-Spectrogram extractor* receives input audio clips and produces corresponding log-mel-scale spectrograms in output.
- *Normalization and Frame generator* executes normalization on all spectrograms extracted and then it produces, from log-mel-spectrogram received in input, $n \times m$ overlapping frames with segmentation. Normalization is done between all spectrograms

with the same ID. For example, in case of min-max normalization, the number of maximum and minimum values is equal to the number of different IDs.

A little digression should be made about what these spectrograms are and how the first block calculate them. A signal is a variation in a certain quantity over time and for audio this quantity is air pressure, sampled by microphones with a particular rate (in the order of kHz). The Fast Fourier Transform (FFT) is an algorithm that allows the decomposition of a signal into its individual frequencies and frequency amplitudes, converting it from the time domain into the frequency domain (generating the spectrum). The Short-Time Fourier Transform (STFT) allows the representation of signals spectrum as they vary over time (generating the spectrogram). As audio signals frequency content varies over time (non-periodic signals), STFT is applied, because it is a FFT computed on overlapping windowed segments of the signal (Figure 3.3).

Spectrograms usually present time on the x-axis, frequencies on the

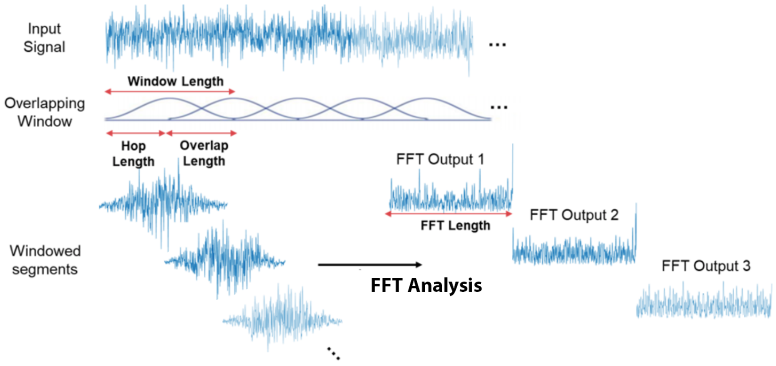


Figure 3.3: Overview of the STFT process.

y-axis and amplitudes are indicated by colors. Log-mel-spectrograms are spectrograms in which the frequencies on y-axis are represented using the mel-scale and the amplitudes are expressed in Decibel (dB). With respect to the normal scale, the mel-scale is a scale of pitches judged by listeners to be equal in distance one from another. The most important parameters involved into this transformation process are the length of the window used for STFT (n_fft), the length of the overlap between two successive windows (hop_length) and the number of bins used for the transformation into the mel scale (n). In the Figure 3.4 a more detailed graphical view of what this block does is illustrated. In the offline phase, $n \times m$ frames are generated from all $n \times q$ spectrograms of the training set, in order to build a new, pre-processed dataset ready for training process.

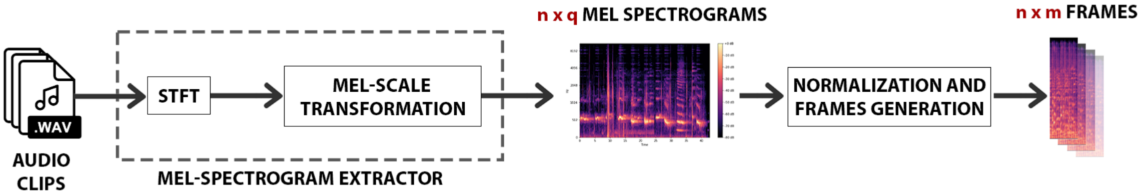


Figure 3.4: Features extraction block

IDs Pre-Processing

This block receives in input a string, the machine ID associated to each audio clip, and converts it to a binary sequence, whose length depends by the number of different versions of M. This is done by the

One-Hot Encoder. One-hot encoding is a technique used to convert some categorical or nominal features in order to make them compatible with training algorithm. Here ID strings are converted. Let's take an example. Supposing to have a machine M and that there are four different versions of it, identified as $ID00$, $ID01$, $ID02$, $ID03$, one-hot encoder converts these strings into $[0, 0, 0, 1]$, $[0, 0, 1, 0]$, $[0, 1, 0, 0]$ and $[1, 0, 0, 0]$, respectively. The length of binary sequences is four because four is the number of versions in this case. This operation arranges the IDs to be inputs of the ID Conditioning Neural Network, better explained in the next sections. In conclusion, since each audio file spectrogram is segmented in frames, this block must ensure that those extracted from the same spectrogram are associated to the same binary sequence.

ID Conditioned Autoencoder

This part of the architecture is obviously the most important one. It receives in input three lists: the list of spectrogram frames, the list of one-hot-encoded ID arrays and the list of strings produced by Label Generator. The latter is used for the loss calculation during the training process, so that it is not a real input of the neural network (the details of what this block does are reported in the next section). In this block there are two main components: the autoencoder, composed by an encoder and a decoder, which receives the spectrogram frames in input, and the ID Conditioning Neural Network, which receives ID binary sequences. Following there is a mathematical representation of these components [14]:

- Encoder $E : \mathcal{X} \rightarrow \mathcal{Z}$ which maps the input X from $\mathcal{X}$ into its encoded version $Z = E(X)$;
- Decoder $D : \mathcal{Z} \rightarrow \mathcal{X}$ which takes the code from $\mathcal{Z}$ and outputs an reconstruction of X;
- Conditioning made by two functions H_γ and $H_\beta : \mathcal{Y} \rightarrow \mathcal{Z}$, taking in input the one-hot encoded ID l from $\mathcal{Y}$ in order to map it into $H_\gamma(l)$ and $H_\beta(l)$, with the same size as code from $\mathcal{Z}$.

The encoder and the decoder can be created with different type of layers, like convolutional, LSTM or fully-connected dense layers. Between the encoder and the decoder there is a latent or encoded representation of the input X, or else Z. Differently from conventional autoencoders, in this architecture the autoencoders are conditional, because decoder input is not Z, but its mathematical combination with the output of conditioning functions:

$$Decoder\ input : H(Z, l) = H_\gamma(l) \cdot Z + H_\beta(l).$$

The conditioning functions $H_\gamma(l)$ and $H_\beta(l)$ can be realized, for example, using dense layers and activation functions. In conclusion, the output of the entire autoencoder is $D(H(E(X), l))$. The goal of ID conditioning is to inform the model about the presence of different machines of the type M, for the recognition of their different normal behaviours. In general, anomalous sound of a machine M with ID x could be similar to normal sound of machine M with ID y and this could generate some false negative (FN). Moreover, normal sound of a machine M with ID w could be different from normal sound of a machine M with ID z and this could generate some false positive (FP). These two situations happen because the autoencoder is unable to separate different machine versions normal behaviour. The key concept is that the autoencoder must be trained to reconstruct normal audio spectrograms placed input only if the provided ID is correct. With this assumptions, after the training, if a normal test sample is placed in input, a low reconstruction error (in terms of mean absolute error or mean squared error) is expected, while if there is an anomalous one, an high reconstruction error is generated, even if this anomalous behavior is similar to a normal behaviour of another machine, thanks to the presence of the ID. The similarity problem is so resolved by the presence of the ID.

Label Generation

At this point a question arises: are the only ID conditioning operations enough for this task? The answer is no because is necessary to edit the training process to help the autoencoder in the recognition of the relationship between IDs and audio clips. For this reason there is the Label Generation block. This block, for each frame, randomly changes with a probability $1-\alpha$ the correct associated ID binary sequence and it adds the string *match* or *not-match* on the basis of decisions. For example if there are 100 audio clips and α is 0.75, 75 audio clips will be associated to the correct ID sequence, the remaining 25 will be associated to a random one, from those available (so that frames belonging to the same spectrogram have the same ID binary sequence and the same string). Moreover, in order obtain what it has been just described, a new loss must be tuned and used in training steps because the classical difference between the encoder input and the decoder output is not enough. In fact, to allow a good reconstruction of the input only when the associated ID is correct, during the training, if the label is *match* the loss function must be calculated in classical way as $||D(H(E(X),l))-X||$, while if the label is *not-match* the loss function must be calculated using an arbitrary value C: $||D(H(E(X),l))-C||$. In conclusion, when the input spectrogram frames are associated to wrong IDs, the autoencoder will generate an high reconstruction error.

3.2.2 Online Operation Phase

The architecture used for online operation phase is very similar to the part just described, except for few components. This part of the architecture is meant for an effective and an operative use on field,

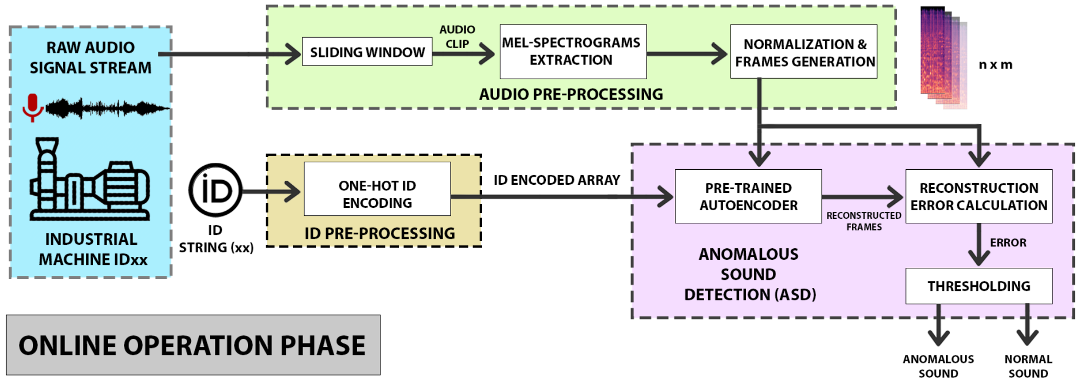


Figure 3.5: Overview of online operating phase of the proposed ASD system.

since the left part of the Figure 3.5 shows that there is not a dataset but a real time raw audio stream from microphones disposed around the machine M. As other elements have been already explored in previous sections, following there is only a description of the new ones and those that are different: Online Audio Pre-Processing block and the Anomalous Sound Detection block.

Online Audio Pre-Processing

It is very similar to the one presented before, with exception of the sliding window component. Because of real-time anomaly detection, there is a continuous audio stream, but the autoencoder model is implemented and trained to work with T seconds audio clips. For this reason, the sliding window component takes in input the stream and samples the last T seconds from it, every h seconds. Next, previously obtained audio clips pass into mel-spectrogram extractor block and into the frame generator, already presented.

Anomalous Sound Detection

This is the final block of the online part of the framework and the one that is responsible of the sound labeling as normal or anomalous. In this part three components are highlighted: the pre-trained autoencoder, the reconstruction error calculator and a threshold. The pre-trained autoencoder is the result of the operations made by the offline part of the architecture. The reconstruction error calculator takes in input the output of the autoencoder and its input in order to evaluate the difference between them. This operation is made for each frame of the spectrogram associated to the audio clip (sampled from the stream) and then an average of recostruction errors is calculated for the comparison with the threshold. In fact, the threshold is used to establish if the average reconstruction error is high enough to classify clips as anomalous. The threshold could be chosen using reconstruction errors of training samples or using different approaches based on ROC curves, like the one presented at the end of Chapter 4.

3.3 Hyperparameters

In order to give a better idea related to the hyperparameters of the proposed model, following there is a summary of the most important ones:

- C: the constant vector that must be reconstructed by the autoencoder when the provided ID is wrong;
- α: the percentage of correct frame-ID couples in training set;
- *NUM_CONDITIONING_LAYERS*: number of layers of the conditioning neural network;
- *NUM_CONDITIONING_NEURONS*: number of neurons used in the the conditioning layers of the neural network and also the dimension of the encoded input generated by the encoder.

Clearly, because of the architecture is compatible with different implementations of autoencoder layers, other hyperparameters that arise from them should be taken in consideration. Moreover, there are other hyperparameters, such as learning rate, batch size and number of epochs, which are not dependent on the proposed architecture but affect the training phase of the model.

3.4 Implementation technologies

In this section, some implementation details about the proposed model are given. The model was implemented in Python 3. Used technologies and libraries are:

- Google Colab[1]: Colab notebooks are Jupyter notebooks that run in the cloud and are highly integrated with Google Drive,

[1] http://colab.research.google.com

making them easy to set up, access, and share;

- Tensorflow[2]: it is an end-to-end open source platform for Machine Learning;
- Keras[3]: it is an open source Deep Learning API written in Python3;
- NumPy[4], used to work in an efficient way with arrays;
- SciKit-Learn[5], a machine learning library, containing simple and efficient tools for predictive data analysis;
- Librosa[6], a Python library for audio and music processing.

[2] https://www.tensorflow.org

[3] https://keras.io

[4] https://numpy.org

[5] https://scikit-learn.org/stable/

[6] https://librosa.org

Chapter 4

Experimental Evaluation

As previously mentioned, this chapter is dedicated to an experimental evaluation of the proposed framework. Firstly, an explanation of how the experiments are conducted is presented, then there is a detailed description of the DCASE dataset, already mentioned in Chapter 2, on which the proposed framework is evaluated. Successively, the chapter shows how data are prepared in order to obtain a proper output from the architecture, with a spotlight on chosen parameters. In conclusion, performance metrics are described and experimental results are reported.

4.1 Experimental protocol

The experimental part of this thesis shows two instances of the proposed framework, each characterized by the way autoencoder is built and using the DCASE dataset for evaluation. In particular, two different autoencoders are taken in consideration: LSTM encoder-decoder and convolutional autoencoder. The purpose of this part is in fact to show the ID conditioning effects as autoencoder layers varies, to demonstrate its compatibility with different encoding and decoding processes. Using the nature of their autoencoder, the overall architectures are so identified as ID Conditioned LSTM Autoencoder (IDC-LSTM-AE) and ID Conditioned Convolutional Autoencoder (IDC-CAE). IDC-LSTM-AE and IDCCAE architectures are implemented and trained on four machines present in DCASE dataset (details reported in following sections).

4.2 Dataset and recording procedure

Previously, in Chapter 2, a brief introduction of DCASE 2020 TASK 2 has been done to introduce some approaches found in literature about the anomaly detection. During the experiments, a part of this dataset is used, in particular the one belonging to MIMII dataset [20]. This dataset contains audio clips recorded from four different machine types: pumps, valves, slide rails and fans. For each machine type there are four different versions. Table 4.1 shows information about machines operations and possible failures that could occur. Clips are recorded by a circular microphone array so that single-channel-based or multi-channel-based approaches can be evaluated. Figure 4.1 depicts the

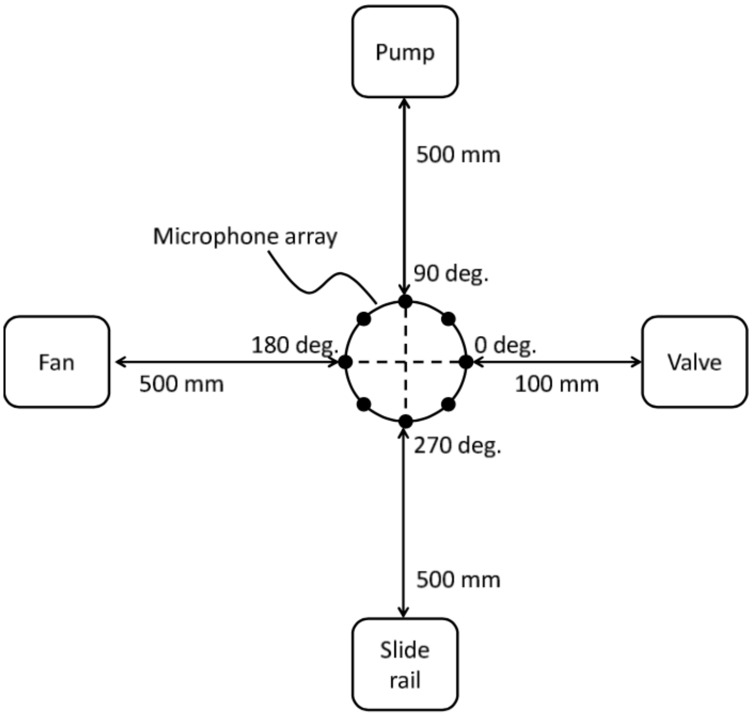


Figure 4.1: Microphones array disposed near machines [7]

recording setup with the direction and distance from each machine type (each machine sound was recorded in separate sessions). To simplify the task, DCASE authors used only the first channel of multi-channel recordings and the sampling rate of all signals has been downsampled to 16 kHz. Last important thing to be noticed is the presence of real

Machine Type	Operations	Some Anomalous Conditions
Pump	Suction from/ discharge to a water pool	Leakage, contamination, clogging, etc.
Fan	It works to provide a continuous flow or gas of air in factories	Unbalanced, voltage change, clogging, etc.
Slide rail	Slide repeat at different speeds	Rail damage, loose belt, no grease, etc.
Valve	Open/close repeat with different timing	More than two kinds of contamination

Table 4.1: Machine descriptions with some anomalous conditions.

factory environmental background noise mixed with the target machines sounds.

DCASE 2020 authors take this dataset and arrange it in order to create *development*, *additional training* and *evaluation* datasets. The first one contains all the necessary for autoencoder training, such as a training set and a test set, while the last one contains test data without condition labels, used for competition submissions. The development dataset is the one used for experiments presented in next sections. Table 4.2 reports machines and the number of audio clips found in training and test sets.

	ID00		**ID02**		**ID04**		**ID06**	
Machine Type	train	test	train	test	train	test	train	test
PUMP	906	243	905	211	602	200	936	202
FAN	911	507	916	549	933	448	915	461
SLIDER	968	456	968	367	434	278	434	189
VALVE	891	219	608	220	900	220	892	220

Table 4.2: Number of training and test samples.

In conclusion, for both instances, four models has been trained, one for each machine type, using training and test sets of all available IDs.

4.3 Pre-Processing Phase

In Chapter 3, mel-spectrogram extractor and all pre-processing blocks have been well described. The goal of this section is to show their details from the experimental point of view. In particular, settings and decisions made for parameters selection regarding mel-spectrogram extraction, normalization, frame generation and IDs pre-processing are presented. Regarding mel-spectrogram extraction, the same parameters are used for all machines: the number of bins (n_mels) is 128, the STFT window (n_fft) is 1024 and the hop_length is 512. Using Librosa functions *load* and *melspectrogram*, each audio clip is firstly converted into a Numpy ndarray representing the signal, successively it is converted to a mel-spectrogram and then the results are added to a final structure. The final structure contains all the extracted mel-spectrograms needed for training. Because of the duration of each clip (10 seconds) and the just mentioned parameters, each mel-spectrogram has the dimension of 128x313. Frame generation specs are reported in Table 4.3.

	IDCCAE		**IDC-LSTM-AE**	
Machine	**Num. Frames**	**Hop-Size**	**Num. Frames**	**Hop-Size**
Pump	15	20	12	25
Fan	15	20	12	25
Valve	15	20	16	18
Slider	21	14	22	13

Table 4.3: Frame generation details.

In particular, the column *Num. Frames* report, for each machine, the number of frames extracted from each spectrogram, while *Hop-Size* indicates the segmentation time-window shift to extract them. Regarding the normalization, a Z-Score is applied on spectrograms sets extracted for each ID of a particular type, before the frame generation. For both instances of the framework, frames placed in input to autoencoders for training have the size of 128x32. In conclusion, regarding the IDs Pre-processing, the ID strings used to identify each machine, regardless the type, are 00, 02, 04 and 06 (as can be seen in Table 4.2) and they are converted respectively to $[0, 0, 0, 1]$, $[0, 0, 1, 0]$, $[0, 1, 0, 0]$ and $[1, 0, 0, 0]$. Definitely, four models per architecture type must be trained to detect eventual anomalies. Moreover, for *match* and *not-match* transformations an $\alpha = 0.75$ is chosen, while the vector C is chosen equal to 5.

4.4 Autoencoder Structure

In this section, the structures of IDCCAE and IDC-LSTM-AE are described. In both instances, the conditioning is a sequence of mathematical operations, in which encoder output and ID conditioning network outputs are involved, as seen in Chapter 3.

The encoded ID is passed through a dense layer and an activation layer to produce the ID conditioning network first output, which is multiplied with encoder output. The second output is the output of

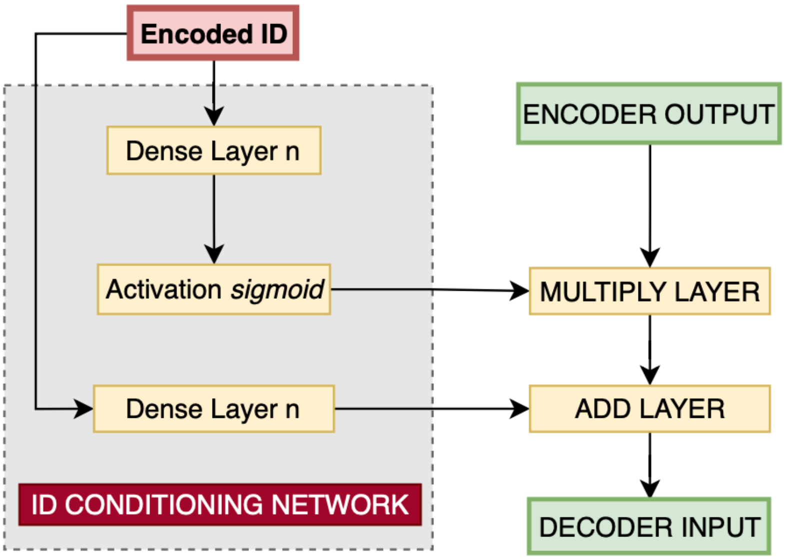


Figure 4.2: Conditioning operations. n is the number of latent variables.

another dense layer with the same encoded ID provided in input. To produce the final representation, which is decoder input, the second output is added to the result of the multiplication. Figure 4.2 better explains how the conditioning is done.

4.4.1 IDCCAE

The Figure 4.3 reports a detailed view of how encoder and decoder block are composed in the convolutional instance of the framework. The encoder network consists in a stack of five hidden layers with convolutional filters of 32, 64, 128, 256, and 512. In particular, as can be seen at the right part of the image, each component of the encoder is a block composed by a stack of different layers: a convolutional layer, followed by batch normalization and the ReLU activation function.

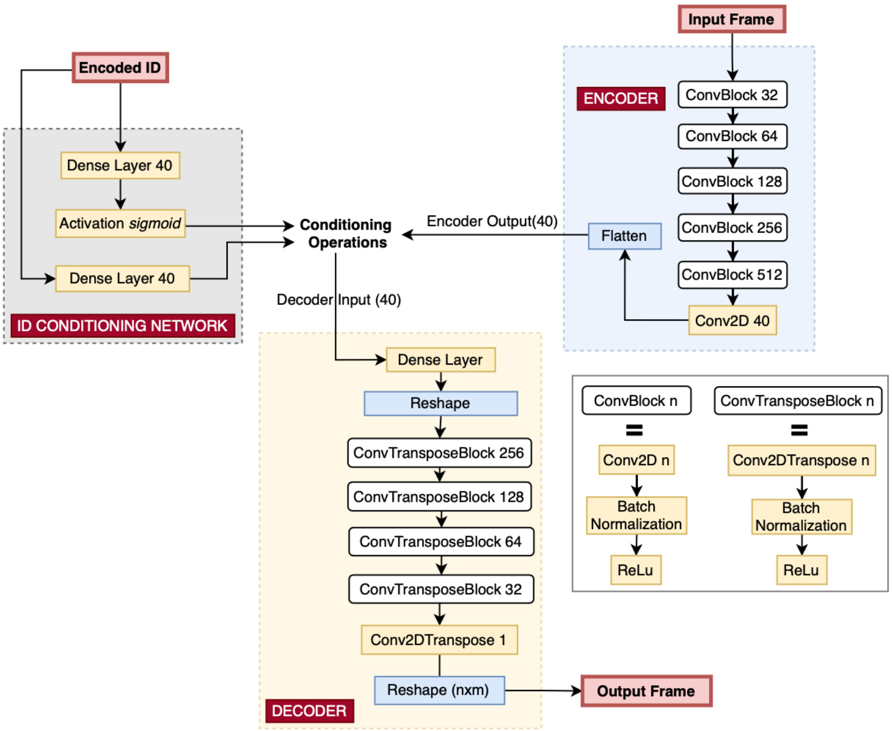


Figure 4.3: ID Conditioned Convolutional Autoencoder

The bottleneck consists of a layer with 40 convolutional filters, reducing the encoder feature maps to a 40-dimensional encoded representation of the input. Regarding the decoder network, first a fully-connected layer reshapes its input to the shape of the last layer of the encoder and then five ConvTransposeBlock (Conv2DTranspose layers followed by batch normalization layers and ReLu activation functions) mirror the encoder. Conditioning operations are those explained in previous sections.

4.4.2 IDC-LSTM-AE

The Figure 4.4 describes the architecture of the Long-Short Term Memory version of the autoencoder. Here, encoder is composed by

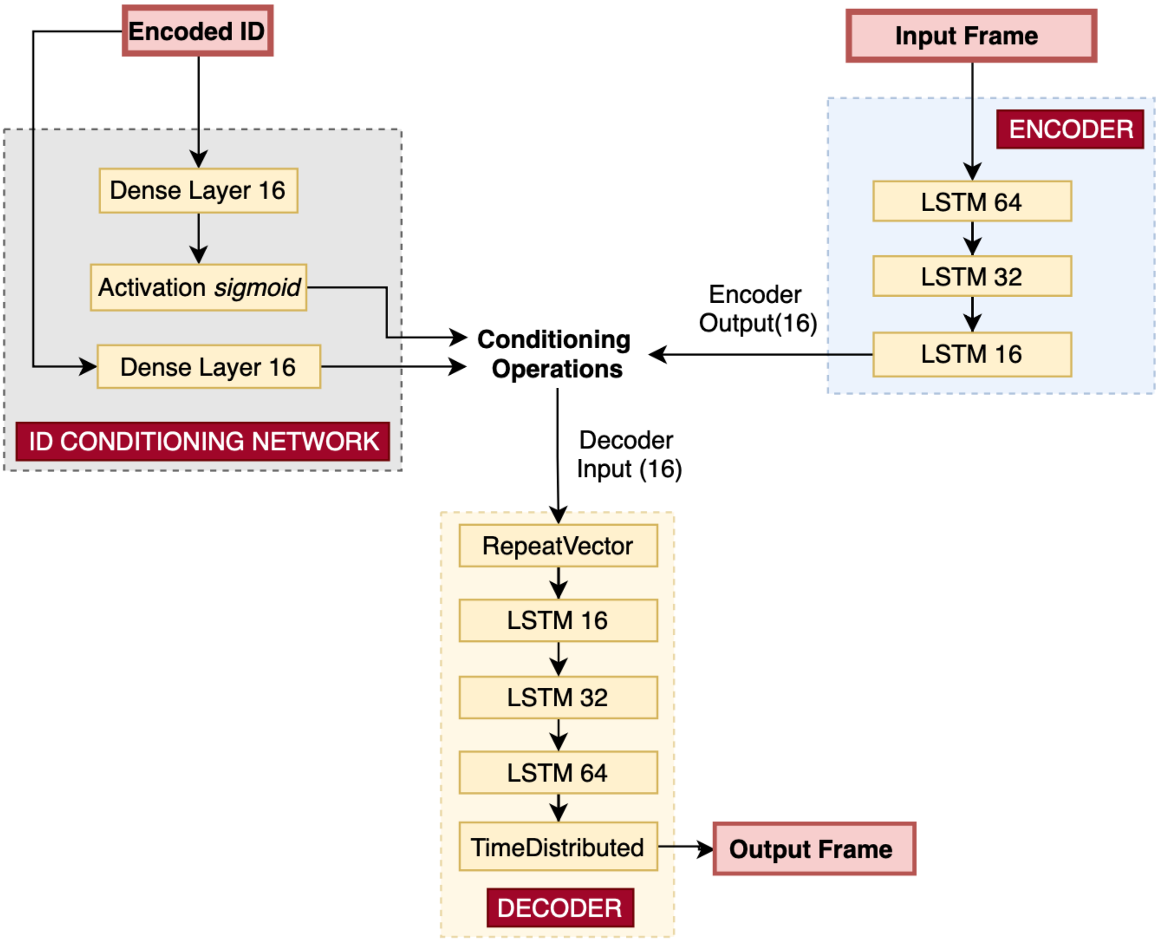


Figure 4.4: ID Conditioned LSTM Autoencoder

three LSTM layers with a decreasing number of units (64,32 and 16), which indicate the dimensionality of their output space. In this architecture, the 128x32 frames placed in input are seen as time-series of 32 timesteps, each characterized by 128 features, which are the frequency amplitudes (the n_mels bins). The decoder is the reversed version of the encoder, but at the beginning there is a RepeatVector layer, which repeats its input n times. In this case the input is repeated 32 times, such as the number of timesteps. This architecture tries to capture the temporal relationship between sequential frequency amplitudes through time, in order to learn a better function to reconstruct the inputs. In this case, encoder output is a 16-dimensional representation of its input, while the conditioning operations are, again, those explained before.

4.5 Other Hyperparameters

In previous sections, autoencoders used to build the two instances of the proposed framework have been visualized, with an explanation of how layers are built. Obviously, during the training phase other parameters are involved, which should be optimized to get as high as possible performances. In the experimental phase of the study, in addition to those seen for IDCCAE and IDC-LSTM-AE, batch size, number of epochs and learning rate are autoencoder-independent hyperparameters taken in consideration during the tuning process. Regarding batch-size, the values $\{64, 128, 256, 512\}$ have been attempted, while for the initial learning rate choice, values like $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ are used. Number of epochs has been varied between 50 and 200. Adam is used as optimizer for trainings.
The Table 4.4 reports the best hyperparameters found during training process.

Regarding the metrics used for reconstruction errors evaluation, Mean Squared Error (MSE) is chosen for all models.

	IDCCAE			IDC-LSTM-AE		
Machine	**BS**	**EP**	**LR**	**BS**	**EP**	**LR**
Pump	256	100	0.0001	512	100	0.001
Fan	512	100	0.0001	256	100	0.001
Valve	64	100	0.0001	512	100	0.001
Slider	12	100	0.0001	512	100	0.001

Table 4.4: Batch size (BS), learning rate (LR) and number of epochs (EP) chosen. These parameters are used to get final results of the experimental part of this text.

4.6 Evaluation and Performance Metrics

The goal of this section is to explain how the performances of trained models are evaluated. First of all, in both cases the test sets are pre-processed in the same way seen for training sets. Regarding the metrics used for models evaluation during the experiments, the area under the receiver operating characteristic (ROC) curve (AUC) and the partial-AUC (pAUC) are considered. These metrics are those used for the competition ranking, when the challenge was still in progress. Remembering that the ROC curve shows the trend of the true positive rate (TPR) in function of the false positive rate (FPR) at the variation of a parameter (like the threshold used for binary classification tasks), the pAUC is calculated as the AUC over a low FPR range $[0, p]$, with $p = 0.1$. Formulas reported by [7] are:

$$AUC = \frac{1}{N_{-}N_{+}} \sum_{i=1}^{N_{-}} \sum_{j=1}^{N_{+}} \mathcal{H}(A_\theta(x_j^{+}) - A_\theta(x_i^{-}))$$

$$AUC = \frac{1}{\lfloor pN_- \rfloor N_+} \sum_{i=1}^{\lfloor pN_- \rfloor} \sum_{j=1}^{N_+} \mathcal{H}(A_\theta(x_j^+) - A_\theta(x_i^-))$$

where $A_\theta(\cdot)$ is the anomaly score generated by the autoencoder, $\lfloor \cdot \rfloor$ is the flooring function and $\mathcal{H}$ returns 1 when $x > 0$ and 0 otherwise. Here, $\{x_i^-\}_{i=1}^{N_-}$ and $\{x_j^+\}_{j=1}^{N_+}$ are normal and anomalous test samples, respectively, and have been sorted so that their anomaly scores are in descending order. Here, N_- and N_+ are the number of normal and anomalous test samples, respectively. According to the above formulas, anomaly scores of normal test samples are used as the threshold. The anomaly score associated to a test sample is calculated taking the reconstruction errors average over all frames extracted from it and, after the application of normalization, placed in input. The pAUC is defined because it is especially important to increase the TPR under low FPR conditions, in that if an ASD system gives false alerts frequently we cannot trust it.

In conclusion, because the results produced with a GPU are generally non-deterministic, means and standard deviations are calculated from 10 independent trials (training and testing of models). In particular, once trained, a model is evaluated on test sets, generating for each ID AUC and pAUC values. Moreover, mean values of AUC and pAUC are calculated from those obtained for each ID. Mean values are used to calculate mean and standard deviation of independent trials. The next section reports results tables.

4.7 Results

This section compares the results obtained from the training of the two models just described and the results published by DCASE authors, related to the solutions proposed by participants. The IDCCAE architecture is compared with a similar version of the architecture without the ID conditioning mechanism [22]. The IDC-LSTM-AE, in the same way, is evaluated using the results obtained by [12], in which a similar LSTM autoencoder without conditioning is trained. Both are also compared with results generated with baseline model provided by authors. Table 4.5 reports convolutional architectures results. Table 4.6 reports LSTM architectures results.

Model	**Pump**				**Fan**			
	AUC		pAUC		AUC		pAUC	
	Mean	Std.Dev	Mean	Std.Dev	Mean	Std.Dev	Mean	Std.Dev
Baseline	72.89%	0,70%	59.99%	0.77%	65.83%	0.53%	52.45%	0.21%
IDCCAE	**76.63%**	1.87%	**67.90%**	1.87%	**71.05%**	0.72%	**70.33%**	0.55%
CAE [22]	72.07%	-	60.96%	-	66.78%	-	52.63%	-
Model	**Slider**				**Valve**			
	AUC		pAUC		AUC		pAUC	
	Mean	Std.Dev	Mean	Std.Dev	Mean	Std.Dev	Mean	Std.Dev
Baseline	84.76%	0.29%	66.53%	0,62%	66.28%	0.49%	50.98%	0,15%
IDCCAE	90.99%	4.30%	**84.14%**	6.46%	74.73%	5.00%	**61.18%**	5.07%
CAE [22]	**91.77%**	-	76.20%	-	**78.83%**	-	53.10%	-

Table 4.5: Mean and std.dev. of AUC and pAUC for convolutional architectures on 10 independent trials. Results found in [22] are reported for comparison. Best results for each metric are marked in bold.

Analyzing Table 4.5 is evident that IDCCAE, regarding pump and fan, obtains better results for both metrics, while regarding slider and

	Pump				**Fan**			
Model	AUC		pAUC		AUC		pAUC	
	Mean	Std.Dev	Mean	Std.Dev	Mean	Std.Dev	Mean	Std.Dev
Baseline	72.89%	0,70%	59.99%	0.77%	65.83%	0.53%	52.45%	0.21%
IDC-LSTM-AE	**78.29%**	2.21%	**69.67%**	2.44%	**67.66%**	2.29%	**65.83%**	1.12%
LSTM [12]	73.94%	-	61.01%	-	67.32%	-	52.05%	-

	Slider				**Valve**			
Model	AUC		pAUC		AUC		pAUC	
	Mean	Std.Dev	Mean	Std.Dev	Mean	Std.Dev	Mean	Std.Dev
Baseline	84.76%	0.29%	66.53%	0,62%	66.28%	0.49%	50.98%	0,15%
IDC-LSTM-AE	82.62%	1.90%	**74.48%**	2.64%	62.98%	2.99%	**59.71%**	1.53%
LSTM [12]	**84.99%**	-	67.47%	-	**67.82%**	-	51.07%	-

Table 4.6: Mean and std.dev. of AUC and pAUC for convolutional architectures on 10 independent trials. Results found in [12] are reported for comparison. Best results for each metric are marked in bold.

valve it obtains better results especially in terms of pAUC. Table 4.6 reports similar results regarding LSTM based models: again conditioned model achieve the best in terms of both metrics for pump and fan, while for slider and valve there are improvements in terms of pAUC. Remarkable are the improvements obtained with conditional autoencoders in terms of pAUC, which means that there are higher values of TPR with lower values of FPR.

4.8 Threshold definition

In Chapter 3, an online version of the proposed framework is shown. It allows a real-time audio classification to establish if a machine is working in a normal state or not. From a technical point of view, the online architecture uses the autoencoder, trained with offline procedures, to reconstruct the spectrograms placed in input. After a reconstruction error evaluation, on the basis of a threshold, audio clips extracted from the stream are classified. Once obtained reconstruction errors, used as anomaly scores, AUC and pAUC are used to compare different solutions and approaches, but the threshold definition step should be done to commission the anomalous sound detector, because without it models can't classify audio inputs. This section tries to explain a possible way to calculate an optimal threshold, but it is not the only one, since different approaches about threshold definition and its optimization process can be found in literature, but this question is beyond the scope of this text.

To calculate an optimal threshold, the concept of optimal must be examined. It depends on the particular use case taken in exam. In fact, once defined a threshold ϵ and a test set is evaluated, a confusion matrix could be calculated, from which true positive rate (TPR, where positive means anomalous), false positive rate (FPR) and other important measures can be extracted. The threshold goodness is related to the weights and the importance associated to these measures. In anomaly detection task is important to have an high TPR and as low as possible value of FPR. To this purpose, the Youden's index J is defined [3]:

$$J = Sensitivity + Specificity - 1$$

$$Sensitivity = TPR = \frac{TP}{TP + FN}$$

$$Specificity = TNR = \frac{TN}{TN + FP} = 1 - FPR = 1 - \frac{FP}{TN + FP}$$

$$J = TPR + (1 - FPR) - 1 = TPR - FPR$$

The higher J is, better the threshold is, according to this definition of optimum, and to achieve the best, the threshold that corresponds to the max value of J must be found.

Practically, the optimal threshold has been calculated using following steps:

1. Reconstruction errors calculated from test set samples (anomaly scores) are collected.

2. Using scikit-learn function *metrics.roc_curve* FPR, TPR and Thresholds, are calculated and also used to visualize ROC curve. FPR, TPR and Thresholds are three arrays of the same length.

3. The optimal threshold corresponds to the element of Thresholds array at the index on which there is a maximum value of $TPR - FPR$. In other way, a vector of the differences between TPR and FPR can be calculated and then the index of the maximum difference on this vector is the index of the optimal threshold in Threshold array.

Following, Figure 4.5 shows the ROC curves calculated for example from IDCCAE architecture trained on audio clips recorded from pumps. It shows the ROC curves in blue and the bisector lines in red, while black dashed lines indicates the values of J. Moreover, red dot is used to mark the FPR and TPR which correspond to optimal thresholds, also numerically reported.

In conclusion, in online detection phases, two alternatives should be taken in consideration:

- Calculate and use a different threshold for each ID string (or machine kind), even if the model used for prediction is one;
- Calculate and use only one threshold, regardless of the different machine versions (last ROC curve in Figure 4.5).

The first option implies that, during detection, the architecture is able to select the right threshold based on the input.

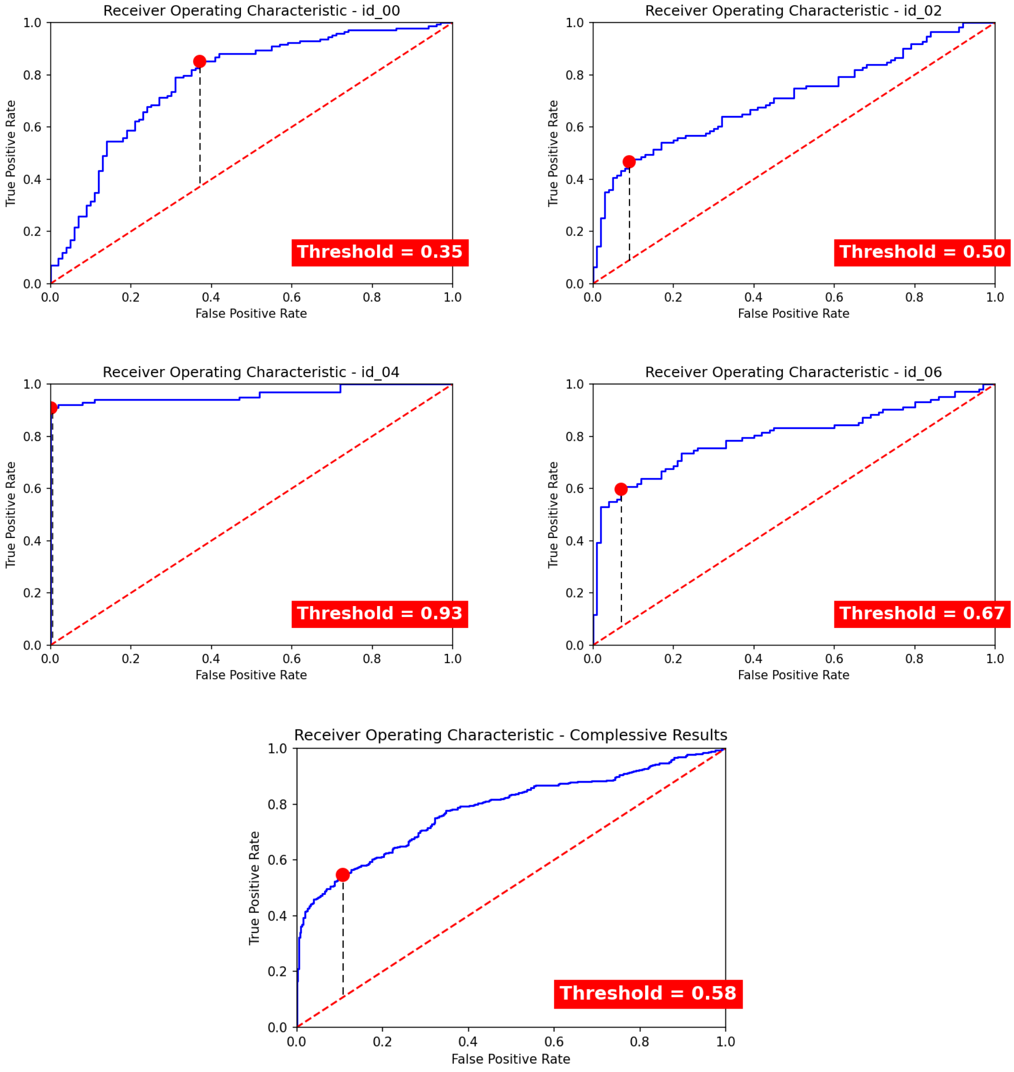


Figure 4.5: ROC curves obtained using IDCCAE model on pumps test audio clips. The image shown below is the ROC curve calculated using the predictions of all four kinds of pump, with the threshold calculated.

Chapter 5

Conclusion

As widely seen, anomaly detection is a data-driven approach for predictive maintenance. Since it gives the possibility to avoid downtime, to reduce costs related to unnecessary operations and to optimize maintenance procedures, in last years a lot of approaches have been developed and studied by researches. In this work, after a state-of-art on autoencoders solutions for anomaly detection task, a conditional autoencoder based framework has been presented to face unsupervised anomalous sound detection. Chapter 3 and Chapter 4 show, respectively, the details of the proposed framework components and experiments performed with it, demonstrating that the involvement of machine identifiers (conditioning) into the autoencoder learning process enables models training on sounds belonging to different machines of the same type (like different pumps), improving their detection capabilities. Moreover, the experimental part of the work shows that the conditioning can be done with a classical neural network, which is also compatible with the various types of encoder-decoder layers. To demonstrate this, experiments have been conducted on DCASE 2020 Task 2 Challenge dataset, which is already arranged to face the task in unsupervised way, replicating a real scenario in which normal sound clips represent most of the available data for training phase. Two instances of the proposed framework have been realized, the first based on convolutional autoencoder, while the second uses a recurrent approach, with LSTM layers in both encoder and decoder. Results have been evaluated using AUC and pAUC, as indicated by the challenge, and they highlight some improvements, with respect to the corresponding non-conditioned autoencoder versions, especially for pAUC.

Literature results and those collected during the experiments are really promising but improvements can still be made. In fact, regarding possible future works, in addition to a more complete hyperparameter optimization, different types of conditioning networks and operations could be attempted, together with the application of some preprocessing strategies to get better training performances, like noise reduction, with the aim of reducing the audio clips background noise surely present in factory environments, or audio data augmentation techniques, like pitching, time-shifting, etc. Obviously, this study could be also extended by the involvement of other, innovative, neural networks, like Variational Autoencoders (VAE) or Generative Adversarial Networks (GAN). Moreover, as the models are trained on audio clips recorded from different machines of the same type, in order to reach better performances, ensemble approaches should be analyzed, considering that there is not an hyperparameter set that is equally good for all of them.

A complexity study could be also very interesting, because in this context, systems computational resources demand should be as lower as possible. In fact, together with accuracy these systems should be also compared in terms of resources used for both training and real time detection, in order to decide which one is better to use. In conclusion, an important consideration to do is that anomaly detection techniques based on audio clips are very innovative in predictive maintenance field. Therefore, is very important to validate various approaches, including the one proposed, in every day real industrial machines working scenarios, in order to study, in the most accurate and truthful way, their possible positive impact in maintenance procedures and costs.

Bibliography

[1] Antonio L. Alfeo, Mario G.C.A. Cimino, Giuseppe Manco, Ettore Ritacco, and Gigliola Vaglini. Using an autoencoder in the design of an anomaly detector for smart manufacturing. *Pattern Recognition Letters*, 136:272–278, 2020.

[2] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. 2020.

[3] Barış Bayram, Taha Berkay Duman, and Gökhan Ince. Real time detection of acoustic anomalies in industrial processes using sequential autoencoders. *Expert Systems*, 38(1), 2021.

[4] Thyago P. Carvalho, Fabrízzio A. A. M. N. Soares, Roberto Vita, Roberto da P. Francisco, João P. Basto, and Symone G. S. Alcalá. A systematic literature review of machine learning methods applied to predictive maintenance. *Computers & Industrial Engineering*, 137:106024, 2019.

[5] Tingting Chen, Xueping Liu, Bizhong Xia, Wei Wang, and Yongzhi Lai. Unsupervised anomaly detection of industrial

robots using sliding-window convolutional variational autoencoder. *IEEE Access*, 8:47072–47081, 2020.

[6] Andrew A. Cook, Göksel Mısırlı, and Zhong Fan. Anomaly detection for iot time-series data: A survey. *IEEE Internet of Things Journal*, 7(7):6481–6494, 2020.

[7] DCASE Coordinators. Unsupervised detection of anomalous sounds for machine condition monitoring.

[8] Ministero dello Sviluppo Economico Italiano (MISE). Nuovo piano nazionale transizione 4.0 - le misure si potenziano e diventano strutturali. 2020.

[9] Yuma Koizumi et al. Description and discussion on dcase2020 challenge task2: Unsupervised anomalous sound detection for machine condition monitoring. 2020.

[10] Tomoki Hayashi, Takenori Yoshimurar, and Yusuke Adachi. Conformer-based id-aware autoencoder for unsupervised anomalous sound detection. 2020.

[11] Ruei-Jie Hsieh, Jerry Chou, and Chih-Hsiang Ho. Unsupervised online anomaly detection on multivariate sensing time series data for smart manufacturing. pages 90–97, 2019.

[12] Anahid Jalali, Alexander Schindler, and Bernhard Haslhofer. Dcase challenge 2020: Unsupervised anomalous sound detection of machinery with deep autoencoders. 2020.

[13] Kamat, Pooja and Sugandhi, Rekha. Anomaly detection for predictive maintenance in industry 4.0- a survey. *E3S Web Conf.*, 170:02007, 2020.

[14] Slawomir Kapka. Id-conditioned auto-encoder for unsupervised anomaly detection. 2020.

[15] Joachim Kirsch, Claus Lintz, Fabian Schmidt, Robert Heiler, and Daniel Heinzler. Anticipate the future - listening to your assets significantly benefits your business. 2019.

[16] Yuma Koizumi, Shin Murata, Noboru Harada, Shoichiro Saito, and Hisashi Uematsu. Sniper: Few-shot learning for anomaly detection to minimize false-negative rate with ensured true-positive rate. pages 915–919, 2019.

[17] Maarten Meire and Peter Karsmakers. Comparison of deep autoencoder architectures for real-time acoustic based anomaly detection in assets. 2:786–790, 2019.

[18] Dong Yul Oh and Il Dong Yun. Residual error based anomaly detection using auto-encoder in smd machine sound. 2018.

[19] YeongHyeon Park and Il Dong Yun. Fast adaptive rnn encoder–decoder for anomaly detection in smd assembly machine. 2018.

[20] Harsh Purohit, Ryo Tanabe, Kenji Ichige, Takashi Endo, Yuki Nikaido, Kaori Suefusa, and Yohei Kawaguchi. Mimii dataset: Sound dataset for malfunctioning industrial machine investigation and inspection. 2019.

[21] Jian Qin, Ying Liu, and Roger Grosvenor. A categorical framework of manufacturing for industry 4.0 and beyond. *Procedia CIRP*, 52:173–178, 2016. The Sixth International Conference on Changeable, Agile, Reconfigurable and Virtual Production (CARV2016).

[22] Alexandrine Ribeiro, Luis Miguel Matos, Pedro José Pereira, Eduardo C. Nune, André L. Ferreira, Paulo Cortez, and André Pilastri. Deep dense and convolutional autoencoders for unsupervised anomaly detection in machine condition sounds. 2020.

[23] Haidong Shao, Hongkai Jiang, Ying Lin, and Xingqiu Li. A novel method for intelligent fault diagnosis of rolling bearings using ensemble deep auto-encoders. *Mechanical Systems and Signal Processing*, 102:278–297, 2018.

[24] Rosaria Silipo, Iris Adä, and Phil Winters. Anomaly detection in predictive maintenance - anomaly detection with time series analysis. 2014.

[25] Rosaria Silipo, Iris Adä, and Phil Winters. Anomaly detection in predictive maintenance - time series prediction, training and prediction without a class. 2018.

[26] Luca Silvestri, Antonio Forcina, Vito Introna, Annalisa Santolamazza, and Vittorio Cesarotti. Maintenance transformation through industry 4.0 technologies: A systematic literature review. *Computers in Industry*, 123:103335, 2020.

[27] Weiting Zhang, Dong Yang, and Hongchao Wang. Data-driven methods for predictive maintenance of industrial equipment: A survey. *IEEE Systems Journal*, 13(3):2213–2227, 2019.