# Using Deep Autoencoders for In-vehicle Audio Anomaly Detection

Pedro José Pereira
*ALGORITMI Centre*
*University of Minho*
Guimarães, Portugal
id6927@alunos.uminho.pt

Gabriel Coelho
*ALGORITMI Centre*
*University of Minho*
Guimarães, Portugal
a82137@alunos.uminho.pt

Alexandrine Ribeiro
*EPMQ - IT*
*CCG ZGDV Institute*
Guimarães, Portugal
alexandrine.ribeiro@ccg.pt

Luís Miguel Matos
*ALGORITMI Centre*
*University of Minho*
Guimarães, Portugal
id6929@alunos.uminho.pt

Eduardo C. Nunes
*ALGORITMI Centre*
*University of Minho*
Guimarães, Portugal
b12176@algoritmi.uminho.pt

André Ferreira
*Bosch Car Multimedia S.A.*
Braga, Portugal
Andre.Ferreira2@pt.bosch.com

André Pilastri
*EPMQ - IT*
*CCG ZGDV Institute*
Guimarães, Portugal
andre.pilastri@ccg.pt

Paulo Cortez
*ALGORITMI Centre*
*Dep. Information Systems*
*University of Minho*
Guimarães, Portugal
pcortez@dsi.uminho.pt

***Abstract*—Current developments on self-driving cars has led to an increasing interest on the business of autonomous shared taxicabs. While most self-driving car technologies focus on the outside environment, there is also a need to provide in-vehicle intelligence (e.g., detect health and safety issues related with the current car occupants). Set within an R&D project focused on in-vehicle cockpit intelligence, the research presented in this paper addresses an unsupervised Acoustic Anomaly Detection (AAD) task. Since data is nonexistent in this domain, we first design an in-vehicle sound event data simulator that includes three types of anomalies: arguing, breaking window and cough. Then, we explore two main sound feature extraction methods (based on a combination of three audio features and mel frequency energy coefficients) and propose a novel Long Short-Term Memory Autoencoder (LSTM-AE) architecture for in-vehicle sound anomaly detection. When considering three synthetic mixture scenarios, competitive results were achieved by the proposed LSTM-AE when compared with two state-of-the-art methods (a dense Autoencoder (AE) and a two-stage clustering).**

***Index Terms*—Anomaly Detection, Audio input representation, Autoencoder, In-vehicle data.**

## I. Introduction

The recent revolution on smart mobility and autonomous cars is producing new businesses and research opportunities, namely in terms of self-driving shared taxicabs. Indeed, there are recent studies addressing this topic, focusing on the car ability to self-drive safely and the possible impacts of self-driving vehicles on society [1]–[3]. A key change of a self-driving shared taxicab is the absence of a designated company driver. Under this context, it is highly relevant to monitor what happens inside the vehicle cockpit, thus developing an automatic computational system that is capable of processing several in-vehicle sensors (e.g., sound, image, air particles) in order to replace the drivers function to monitor and control health, safety and comfort concerns. In particular, such system should be able to detect abnormal events occurring inside the vehicle (e.g., heart attack of a single occupant, fight between two passengers). Understanding these events enables Mobility Service Providers (MSPs) to define possible actions that aim to ensure the occupants' safety and comfort (e.g., calling for medical assistance). Within this context, in this paper we consider audio data, aiming to perform an unsupervised detection of abnormal events based on audio.

In this work, we are interested in a Sound Event Detection (SED) [4] form that involves identification of anomalies from normal audio signals, which is known as Acoustic Anomaly Detection (AAD) [8]. In particular, we focus on an unsupervised AAD, since labeled data is costly, requiring a manual effort that is time consuming and prone to human errors [7]. Over the last years, several studies addressed unsupervised AAD by means of different learning algorithms. Examples include the One-Class SVM (OCSVM) [9], [10] and Isolation Forest (IF) [11], [12] algorithms. A two-step clustering algorithm, named IRESE, was also recently proposed for AAD [7]. This paper focuses more on Autoencoders (AE), a deep learning neural architecture that became a popular for AAD. When compared with OCSVM, IF and IRESE, the AE training is computationally faster, thus it can handle larger amounts of training data.

An AE is composed by an encoder, a latent space and a decoder. The training algorithm attempts to generate output values that are identical to its inputs. When applied to anomaly detection, only normal event data is used during the AE training phase. By learning to reconstruct only normal events, when it tries to reconstruct abnormal unseen data, AEs produces higher reconstructing errors. By assuming a reconstruction error threshold, such AE outputs can be interpreted as anomalous. AE has been widely used in unsupervised AAD tasks [13], [14]. Recently, this standard Deep Learning model was used the baseline for the popular Detection and Classification of Acoustic Scenes and Events (DCASE) challenge task 2 [15]. Different AE architectures have been proposed

for unsupervised AAD, such as Convolutional AE [8]. There are also studies that apply AEs for AAD feature extraction, by using its latent space output, such as proposed in [16] for OCSVM and [12] for IF.

Even though several studies have targeted the unsupervised AAD topic, most of them were applied to industrial processes [8], [14], [15]. There are few studies that addressed SED and AAD within in-vehicle environments by using a supervised learning approach, such as: to classify noise [17]–[19], to estimate vehicle speed and gear position [20] and to detect shouts and screams in public transports [21].

Within our knowledge, there are no research studies and public datasets related with unsupervised in-vehicle AAD. To handle this task, in this work we propose two systems: a synthetic anomalous event simulator and a real-time unsupervised audio anomaly detection system. The former system uses real-world audio data, recorded inside a car cockpit while driving and that is mixed it with different normal and anomalous events (e.g., breaking a window, someone arguing, people talking and texting), aiming to achieve a realistic labeled audio dataset. The latter uses the synthetic mixture sound data to train machine learning algorithms to detect unusual events on data without using any labels. In particular, we propose a novel Long Short-Term Memory Autoencoder (LSTM-AE) neural network architecture, which is compared with a deep dense AE and the IRESE method. The learning methods are analyzed in terms of two main feature extraction methods: a combination of three audio features [7] and usage of mel frequency energy coefficients [15].

The paper is organized as follows. Section II describes the sound anomaly data simulator system and generated datasets, the extracted audio features, the unsupervised learning AAD methods and the evaluation procedure. Computational experiments and respective results are discussed in Section III. Finally, Section IV highlights the main conclusions and future work directions.

## II. Materials and Methods

### A. In-vehicle Sound Data

This research was developed within a larger R&D project focused on general in-vehicle intelligence and that involves the Bosch Car Multimedia S.A. (BCM) company. This paper presents the unsupervised AAD component of such project.

With the purpose of creating rich sound anomaly datasets, we developed a computational system written in the Python language (making use of its `numpy`, `soundfile` and `librosa` modules) that performs an audio generation task that simulates a realistic set of events fo an in-vehicle environment. A set of features of the computational system were inspired on the `TUT` system that was created for the DCASE 2016 competition [22]. Nevertheless, our system includes unique features related with the in-vehicle AAD task. The developed system works as a synthesizer, using background recordings and acoustic scenes relative to a set of predefined anomalous behaviors (rare events). This process comprises several phases, present in Figure 1, namely:

- **Inputs:** the two audio sets to be mixed (in-vehicle background and event sounds) and a third auxiliary file with the mixture parameters (e.g., sample rate).
- **Auxiliary Files:** automatically generated when the inputs are given and containing information related the sound files (e.g., file paths and type of event).
- **Mixture Recipe:** the recipe includes several parameters that control the type of mixture performed, such as the time stamp when the rare event will be inserted inside the background sound.
- **Output:** the final generated audio mixtures. During this process, diverse operations are computed if needed (e.g., resampling, handing the clipping effect). In addition, the tool creates metadata files related with event labels.

In order to generate the sound datasets, it was required to collect audio files related with in-vehicle background and rare-event sounds. The background audios were recorded by elements from the BCM company and are related with real audio recordings of several car driving trips. Due to privacy and commercial issues, these collected recordings are not made publicly available. The collected background audio includes typical in-vehicle driving events, such as: people eating, putting their seat belts, turn signals, vibrations, radio and parking sensors. We defined a total of 194 audio clips related with the "Background" normal R&D project use case. Each clip has a duration of 60 seconds and it was randomly sampled from the whole raw car driving audio data. The background clips were manually labeled, allowing to identify two subclasses: radio on - 160 clips; and radio off - 34 clips.

Regarding the rare event sounds, audio clips with a few seconds (thus shorter than the background clips), were collected from two public sources: Freesound[1] search engine and Librivox[2]. The collected sound events are related with the other 7 use cases from the R&D project: **anomaly events** - people arguing, breaking a window, coughing; and **normal event** - reading (e.g., a book), singing, talking, using a smartphone (e.g., texting). Table I summarizes the retrieved audio events: the type of **Use case**, if it is an anomaly (column **Anom.**) and the total number of audio clips (column **Clips**).

Three datasets were created for the unsupervised AAD experiments. The simulation tool parameters are shown in Table II. The sound mixture tool works by randomly selecting: a background clip (with 60 s), a non background event clip (e.g., arguing) with a much shorter duration and a $t_i$ temporal insertion point ($t_i \in [0.0, 60.0[$), which defines the starting time (e.g., $t_i = 15.45$ s) where the non background clip is inserted and merged into the background.

The first parameter (`seed`) defines the random initial seed. The second parameter `event_presence_prob` sets the probability that a particular non background event will be selected for the mixture. The default value of 1 means that all 7 non background events are included. The third parameter (`mixtures_per_class`) sets the amount of clips randomly

[1]https://freesound.org/

[2]https://librivox.org/

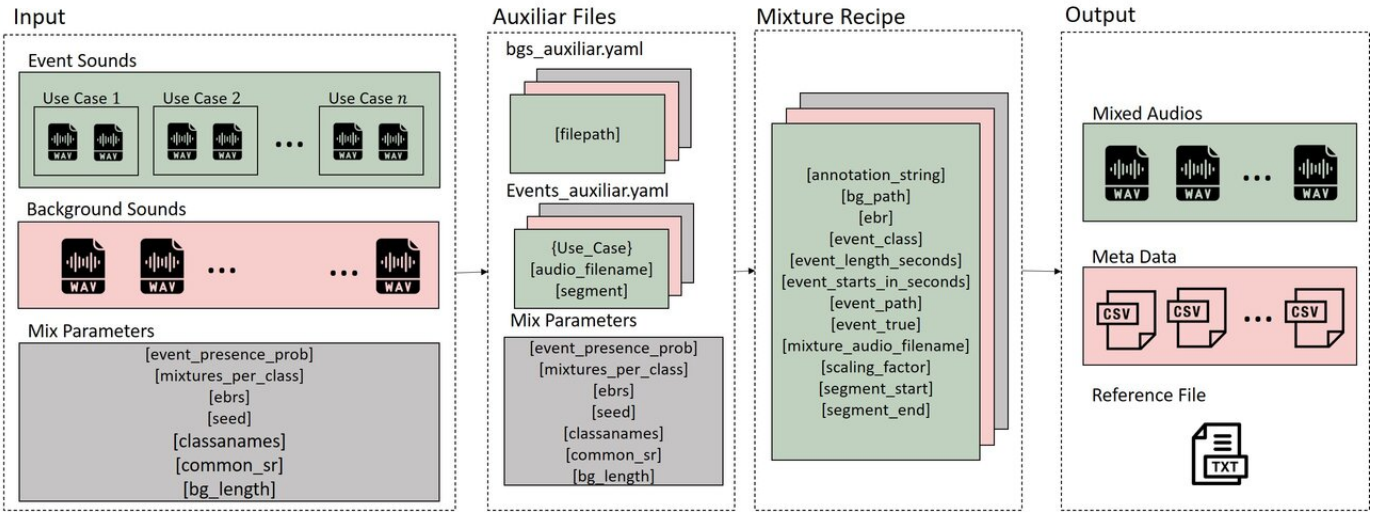


Fig. 1: Worflow of the developed audio simulation tool.

TABLE I: Audio Data Description.

| Use case | Anom. | Clips | Split | Class | Mix3 (s) | Mix6 (s) | Mix6NR (s) |
|---|---|---|---|---|---|---|---|
| Arguing | ✓ | 32 | train | normal | 3443 | 3469 | 3521 |
| | | | | anomaly | 757 | 731 | 679 |
| | | | test | normal | 1460 | 1434 | 1470 |
| | | | | anomaly | 340 | 366 | 330 |
| Breaking Window | ✓ | 64 | train | normal | 3572 | 3577 | 3787 |
| | | | | anomaly | 628 | 623 | 413 |
| | | | test | normal | 1546 | 1541 | 1530 |
| | | | | anomaly | 254 | 259 | 270 |
| Cough | ✓ | 49 | train | normal | 3774 | 3807 | 3888 |
| | | | | anomaly | 426 | 393 | 312 |
| | | | test | normal | 1671 | 1638 | 1670 |
| | | | | anomaly | 129 | 162 | 130 |
| Background | | 194 | train | normal | 4200 | 4200 | 4200 |
| | | | test | normal | 1800 | 1800 | 1800 |
| Reading | | 67 | train | normal | 4200 | 4200 | 4200 |
| | | | test | normal | 1800 | 1800 | 1800 |
| Singing | | 47 | train | normal | 4200 | 4200 | 4200 |
| | | | test | normal | 1800 | 1800 | 1800 |
| Talking | | 71 | train | normal | 4200 | 4200 | 4200 |
| | | | test | normal | 1800 | 1800 | 1800 |
| Using Smartphone | | 13 | train | normal | 4200 | 4200 | 4200 |
| | | | test | normal | 1800 | 1800 | 1800 |

selected for each event type. The value of 100 means that the final mixture contains a total of 8 (events) $\times$100 = 800 minutes of sound (with 100 minutes of background clips, 100 minutes of background merged with one arguing event, and so on). Each event clip is randomly selected. When the collected number of events is larger than `mixtures_per_class` (e.g., background case), then an undersampling procedure is applied. Similarly, when there are less event clips (e.g., Arguing), then an oversampling is executed. The fourth parameter `ebrs`=$(s_l, s_m, s_h)$ is related with the background ratio sampling procedure. This procedure assumes that an inserted sound clip can have (with the same probability) a lower ($s_l$), identical ($s_m$=0) or higher signal ($s_h$) when compared with the background sound strength, with $s_l < 0$ and $s_h > 0$ denoting the sound signal strength changes in dB. The goal is to mimic different locations for the non background events, simulating as if they were generated at a normal distance ($s_m$), further away ($s_l$) or closer ($s_h$) to the sound microphone. As for the fifth parameter (`common_sr`), it defines the final sample rate of the generated sound mixture (in Hz).

TABLE II: Summary of the generated sound ADD dataset parameters.

| | | Dataset Name | | |
|---|---|---|---|---|
| | | Mix3 | Mix6 | Mix6NR |
| **Mixture Parameters** | `seed` | 50 | 50 | 50 |
| | `event_presence_prob` | 1 | 1 | 1 |
| | `mixtures_per_class` | 100 | 100 | 100 |
| | `ebrs` | {-3, 0, 3} | {-6, 0, 6} | {-6, 0, 6} |
| | `common_sr` (Hz) | 44100 | 44100 | 44100 |
| **Background audios** | radio on | ✓ | ✓ | – |
| | radio off | ✓ | ✓ | ✓ |

The first two datasets include both types of background sounds (with radio on and off, sampled from all 194 clips) and are distinguished in terms of the `ebrs` value. Mix3 assumes a smaller in-vehicle size (corresponding to the lower absolute $s_l$ and $s_h$ values), while Mix6 mimics a larger vehicle interior size (e.g., van). The third dataset (Mix6NR) only uses radio off background clips (sampled from the 34 audio clips), thus it should be easier for AAD when compared with Mix6. The three datasets include total of 800 minutes (48,000 s), which were split into training (with 70% of the clips) and test sets (with 30% of the clips) by using a stratified random sampling. Table I presents the length (in s) of train and test sets when associated with the event type for the three datasets. Since train and test split was based on the one minute sound

segments, and not on event duration, there is a slight difference on duration values for each mixture. In the table, the length associated with anomaly events (e.g., arguing) corresponds to the background segments that include such anomaly events, thus the sound duration is subdivided in terms of the "normal" and "anomaly" classes, where "normal" denotes the normal time of the changed background. It should be noted that all three datasets are unbalanced, including a larger duration of normal sounds, which is similar to what would occur in real environment. The unbalanced distribution is due to two reasons: there are more normal events than anomalous ones; and the background segments with an anomaly have still a larger length of "normal" background sound.

The merge of sound files is based on the sum of audio signals. The clipping phenomenon occurs when the resulting audio signal is amplified beyond its output capabilities. To handle this issue, the mixture tool uses an anti-clipping factor of 0.2 that is multiplied over the resulting audio [22]. Then, the `librosa` Python module is used to normalize the audio [23], using the maximum absolute value as norm, in order to maintain it audible. After generating the sound mixtures, all audio files were manually listened to ensure that realistic scenarios were created. The final generated audio file includes metadata with the file name and path, duration, starting and ending event time stamps and its label (normal or anomaly).

### *B. Feature Extraction*

We explore two main methods to perform the audio feature extraction based on the [7] and [15] state-of-the-art AAD frameworks. Both frameworks assume that the model handles only one second of input data when attempting to detect an anomaly. The two feature extraction methods were implemented in Python by using its `librosa` module [23].

The first method, denoted here as **MGL**, is based on IRESE framework [7] and it extracts three types of features from an audio sample: **M**el Frequency Cepstral Coefficients (MFCCs), **G**ammatone Frequency Cepstral Coefficients (GFCCs) and **L**inear Predictive Coding coefficients (LPCs). When applied to one second of 44100 Hz data, the method initially computes a total of 40 MFCC coefficients for 40 audio sample frames, 40 GFCC values for 274 audio samples and a total of 10 LPC elements. Then, the distinct audio sample values (for MFCC and GFCC) are aggregated by using an averaging function. Thus, the final input vector contains a total of 90 numeric features, as shown in Figure 2.

The second method is based on the Mel Frequency Energy Coefficient (**MFEC**), which is a log mel-band energy. It is important to note that since each AAD algorithm has its own particularities, we have defined different MFEC extraction procedures (as shown in Figure 3). For the dense AE, the feature extraction setup is identical to the DCASE challenge task 2 [15]. Firstly, the MFEC values are computed by using a Fast Fourier Transform (FFT) with a window size of 1024. For the one second 44100 Hz data, this produces several 23 ms frames with a 50% overlap, where 128 MFECs are extracted for each frame. Then, a context window of size 5

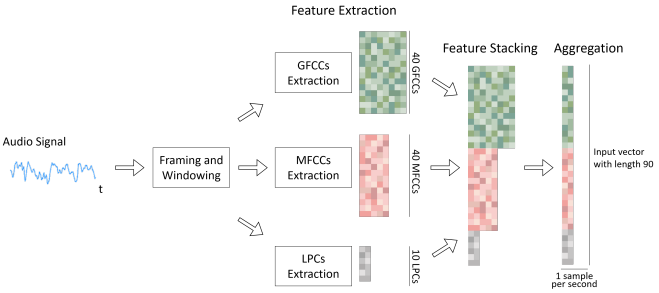


Fig. 2: The **MGL** feature extraction method.

is used (with 2 previous frames, the current frame and the next 2 frames), resulting in an input row vector with a 5×128 = 640 values. Overall, one second produces a total of 86 input row vectors (with 86×640= 55,040 numeric values). The same approach was tested with the clustering IRESE method, however due to the high dimensionality of the input data, the computational effort was too high to be processed in a reasonable amount of time. To solve this issue, we adopted the averaging aggregation used by the MGL method, computing the average row value (total of 640 inputs). Regarding the LSTM-AE, the LSTM component requires the temporalization of the data. Therefore, we adapted the feature extraction after the computation of the 128 MFECs for each frame. Using preliminary AAD experiments with the DCASE challenge task 2 data (related with detecting mechanical failures in working machines), a time step with 8 consecutive frames (each with 128 MFECs) was defined. Then, a sliding time window is applied to the one second sound segment, with the fixed length size of 8×0.023 = 0.184s and an iteration jump of 1 frame (23 ms). This results in a 3-dimensional array with $N = 43$ (sliding window sample iterations) × 8 (time steps) × 128 (MFECs), corresponding to a total of 44,032 input features. It should be noted that for the dense and LSTM AEs, a large input space is defined (with 55,040 and 44,032 inputs), which is similar to the raw audio data (44,100 values per second). However, in preliminary experiments (using DCASE task 2 data) we have obtained worst AAD results when adopting raw input audio data instead of the **MFEC** method to feed the AEs.

### *C. Autoencoders*

To build an AE-based AAD, the AE is trained only with normal event data. Part of the training data (10% of the data, randomly sampled and not used to fit the AE models) is used as validation set. The reconstruction error computed on this validation set is used to tune the hyperparameters and perform an early stopping training. Then, in the inference phase, new data is fed to the AE for reconstruction. The reconstruction error ($r_s$ for sound sample $s$) is used as the classifier decision score, with a higher value corresponding to a higher anomaly probability. Thus, a sound sample $s$ is considered to be anomalous when the AE reconstruct error

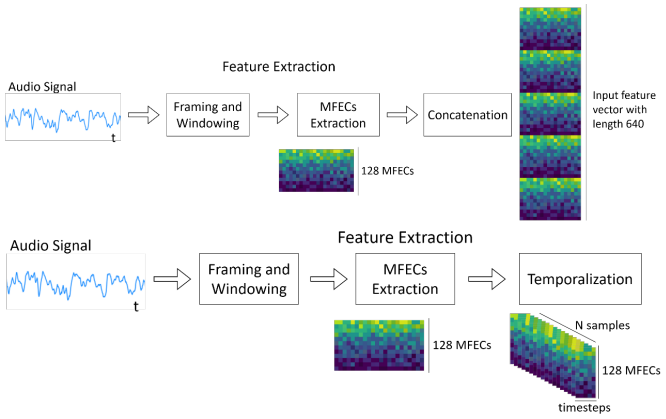


Fig. 3: The **MFEC** feature extraction method for the dense AE and two-step clustering (top) and AE-LSTM (bottom) algorithms.

exceeds a predefined threshold value ($r_s > Th$). The overall representation of such system is shown in Figure 4.

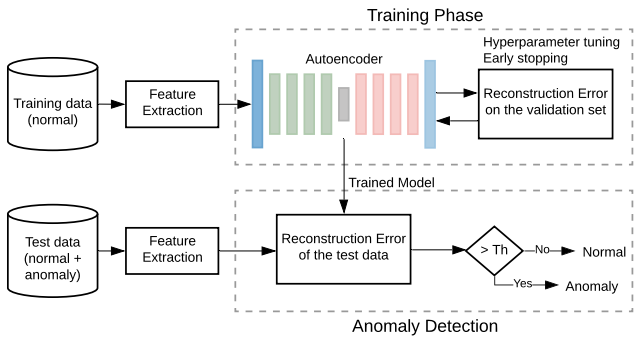


Fig. 4: Schematic of the Autoencoder anomaly detection system.

The computation of the Receiver Operating Characteristic (ROC) curves requires only target test labels and decision scores. However, in a real in-vehicle intelligence deployment scenario, a fixed $Th$ value would be needed to interpret the reconstruction values and assign class labels. In such scenario, a validation set with both class labels (normal and anomaly events) can be used to select the $Th$ value that provides the best trade-off between specificity and sensitivity.

In the scope of this work, we explore two AEs, dense and LSTM based, to perform AAD. The training data for the AEs correspond to the audio recordings with only normal events, whereas the test data contains audio with all event types (normal and anomalies). The AEs were implemented by using the `keras` Python module [24], assuming the Adam training algorithm, with a maximum of 100 epochs, early stopping and Mean Squared Error (MSE) loss function. The MSE measure is also used as the reconstruction error metric.

The dense AE assumes a deep fully-connected AE architecture that includes a stack of four layers with the same number of hidden nodes ($L_h$) in both its encoder and decoder components, training with batch normalization and ReLU activation function (top of Figure 5) [15]. Let $(I, L_h, ..., L_h, L_b, L_h, ..., L_h, I)$ denote the AE neural structure, where $I$ represents the number of inputs and $L_b$ the bottleneck layer size. The original DCASE baseline model used $L_h = 128$ and $L_b = 8$ hidden nodes (defining the latent space). However, since the model was proposed for a different sampling rate (16,000 Hz), we performed some preliminary experiments with our in-vehicle 44,000 Hz data, in which we selected the lowest reconstruction validation error when exploring distinct hidden node values ($L_h \in \{64, 128, 512, 1024\}$). The best results were achieved for $L_h = 512$ and thus the tested dense AE uses the structure $(I, 512, 512, 512, 512, 8, 512, 512, 512, 512, I)$.

Since LSTM tend to provide good results when modeling sequential data, we propose an LSTM-AE architecture for AAD (bottom of Figure 5). In this AE, the encoder and decoder components are composed of LSTM layers. Similarly to the simpler AE, each component is composed of a stack of four LSTM layers with $L_h$ units. There is also a bottleneck dense layer with $L_b$ hidden nodes, where $L_b < L_h$. This layer is preceded by a LSTM layer with the same size ($L_b$), which acts as a bridge, preparing the data for the dense layer. Since the decoder network is designed to unfold the encoding component, the decoder layers were stacked in the reverse order of the encoder ones. Given that the proposed LSTM-AE is new (and not used in the DCASE competition), preliminary experiments were also conducted to tune both $L_h$ and $L_b$ values by using a grid search with power of 2 values ($L_h \in \{64, 128, 512\}$ and $L_b \in \{4, 8, 16, 32\}$). The best validation error results were achieved with $L_h = 128$ and $L_b = 16$ hidden units. Thus, the tested LSTM-AE assumes the structure $(I, 128, 128, ..., 16, 16, 16, 128, ..., 128, I)$.

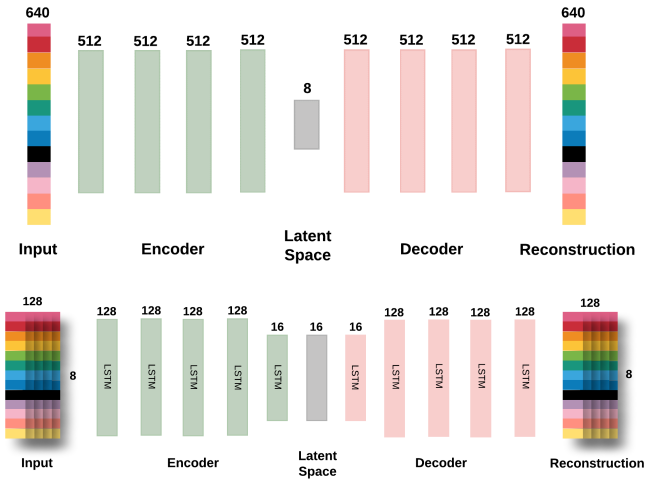


Fig. 5: Dense-AE (top) and LSTM-AE (bottom) architectures.

### D. Two-stage Clustering

For comparison purposes, we selected the recent IRESE two-stage clustering approach [7]. The IRESE training data

contains both normal and anomaly events. Considering that anomaly records should be different from normal ones, the method partitions the data into two main clusters. In the first stage, IRESE uses an online BIRCH micro-clustering algorithm to extract relevant intermediate features from the input data, forming a set of micro-clusters. In a second stage, an offline Agglomerative macro-clustering (with Ward linkage criterion) is executed, aggregating the micro-clusters into the two main clusters. Since data labels are not used and assuming that normal events are far more frequent than anomalies, events belonging to the cluster with more records are considered to be normal and the remaining ones anomalies. The overall IRESE procedure is presented in Figure 6.

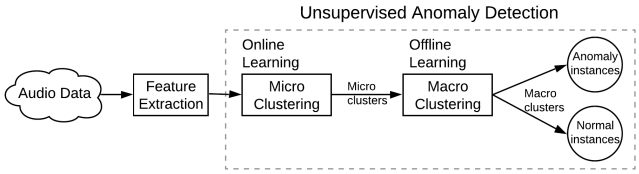


Fig. 6: The IRESE AAD approach.

The original IRESE approach only outputs class labels, which are assigned by selecting the macro-cluster that contains the micro-cluster with the shortest Euclidean distance to a given sample $s$. In order to compute the ROC curve [25], a decision score is required. In this work, we define such anomaly score ($a_s$) as: $a_s = K + \min(d_{0,s}) - \min(d_{1,s})$, where $d_{0,i}$ and $d_{1,s}$ denote two sets with all Euclidean distances from sample $s$ to the normal (0) and anomaly (1) micro-clusters (obtained using training data). The rational is to increase the anomaly score if the distance to the normal macro-cluster increases or if the distance to the anomaly macro-cluster decreases. To avoid negative $a_s$ scores, the constant $K$ is set as the absolute value of the maximum $a_s$ score when $K = 0$ for all tested instances. Similarly to the AEs, a threshold is used to assign the class labels, where an anomaly is considered true if $a_s > Th$. If needed, the $Th$ explicit value can be set by using the same procedure adopted for the AEs (Section II-C). This two-stage clustering learning method was implemented using the `scikit-learn` Python module [26].

### *E. Evaluation*

To evaluate the methods we adopted the ROC curve and two popular AAD metrics [15]: the Area Under the ROC Curve (AUC) and the partial-AUC (pAUC). The ROC curve shows the False Positive Rate (FPR) or one minus the specificity ($x$-axis) versus the True Positive Rate (TPR) or sensitivity ($y$-axis), for all possible threshold values [25]. The overall discriminatory performance is given by the area under the curve ($AUC = \int_0^1 ROC dK$). The AUC metric has two advantages [27]: firstly, quality values are not affected if the classification data is unbalanced (which is our case); secondly, the values have a easy human interpretation (50% performance of a random classifier; 60% - reasonable; 70% - good; 80% - very good; 90% - excellent; and 100% - perfect). As for pAUC, it computes the AUC for a particular range of interest. For the AAD task, we assume the same FPR $[0, 0.1]$ range used in [15], which favors a more specific model (with less false alarms).

As explained in Section II-A, the three audio mixture datasets include a total of 800 audio segments, each with one minute of an in-vehicle driving background sound, which can contain a random merge of one normal (e.g., someone talking) or abnormal (e.g., cough) event. These segments were split into training (70%, 560 segments) and test (30%, 240 segments) audio files. Given that the AAD models work on a one second basis (Section II-B), each sound second corresponds to an instance, thus all classification metrics were computed for each second/instance, with the training sets including a maximum of 33,600 instances (only normal instances are used to fit the AEs, thus the training set size is smaller for such models) and the test sets a total of 14,000 examples (with both normal and anomaly events). All results reported in the Section III were computed using the test set values.

## III. Results

The experiments were executed in an Intel Xeon 1.70GHz server. For each AAD method, the training time (in s) and average prediction time to handle one second of sound (in ms) were recorded. The results are presented in Table III.

TABLE III: Comparison of test set results for different audio mixtures (best results per mixture in **bold**).

| Audio Mixture | Sound Features | AAD Model | Train Time (s) | Predict Time (ms) | AUC | pAUC |
|---|---|---|---|---|---|---|
| Mix3 | **MGL** | AE | 45 | 1.0 | 70.75 | 57.16 |
| | | LSTM-AE | 20 | 1.5 | 51.05 | 52.41 |
| | | IRESE | 8733 | 4.0 | 45.59 | 47.71 |
| | **MFEC** | AE | 4861 | 0.7 | 73.08 | 56.87 |
| | | LSTM-AE | 8243 | 25.7 | **77.27** | **60.29** |
| | | IRESE | 12365 | 4.1 | 46.12 | 47.71 |
| Mix6 | **MGL** | AE | 53 | 1.1 | 68.26 | 53.52 |
| | | LSTM-AE | 19 | 0.3 | 52.61 | 51.71 |
| | | IRESE | 8194 | 4.0 | 45.16 | 47.47 |
| | **MFEC** | AE | 3564 | 0.8 | 72.21 | 56.96 |
| | | LSTM-AE | 8015 | 25.7 | **78.27** | **60.81** |
| | | IRESE | 12292 | 4.1 | 46.35 | 47.50 |
| Mix6NR | **MGL** | AE | 46 | 0.1 | 75.01 | 58.27 |
| | | LSTM-AE | 25 | 0.2 | 58.87 | 53.76 |
| | | IRESE | 11641 | 4.0 | 41.24 | 48.11 |
| | **MFEC** | AE | 1010 | 6.0 | 77.56 | 57.38 |
| | | LSTM-AE | 8293 | 31.1 | **78.54** | **58.57** |
| | | IRESE | 23129 | 4.0 | 40.36 | 48.42 |

The IRESE baseline provided the worst computational effort and classification results. In particular, a poor discrimination (lower than a random classifier) was achieved. To understand these results, we measured the original IRESE class label assigned method over the test data and found that the method tends to produce very low TPR values (e.g., it is 8% for Mix3). Thus, the poor AUC results are justified by the large number

of anomaly events are included in the normal macro-cluster, which prejudices the quality of the anomaly decision score computation.

When analyzing the AEs, the **MFEC** feature extraction requires a larger training computational effort (since it uses much more inputs) but it consistently provides the better AUC values when compared with **MGL**. Overall, the **MFEC** LSTM-AE approach provides the best results (for both AUC and pAUC metrics) for the three mixtures. In particular, interesting AUC values (close to 80%) were achieved: 77% for Mix3, 78% for Mix6 and 79% for Mix6NR. Indeed, the best LSTM-AE approach outperforms the simpler AE by 4 (Mix3), 6 (Mix6) and 1 (Mix6NR) percentage points. It should be noted that the largest training times (8,293 s for **MFEC** LTSM-AE) are still shorter than the training sound file (with 33,600 s), thus the training models can be achieved in real-time. Also importantly, the average response time is very fast, related with a few ms and thus applicable in a real in-vehicle setting.

To further inspect the quality of the selected model (**MFEC** LSTM-AE), we have filtered the test sets of the three mixtures. Each filtered subset includes only the one minute audio background segments that have one of the anomaly use cases from Table I: Arguing, Breaking Window and Cough. Thus, the three test subsets contain a total of 30 (segments) $\times 60$ (s)= 1,800 instances. The goal is to access the LSTM-AE model quality to correctly identify an anomaly (the positive class) when it appears in a vehicle trip. Table IV shows the respective classification performance metrics (AUC, pAUC and also TPR) that were computed using these filtered test sets. For all mixtures, the best classification performance was obtained for the Arguing event, followed by the Cough and then Breaking Window. In general, high quality metric values were obtained, with the AUC values ranging from good (71% - Breaking Window and Mix3) to excellent (94% - Arguing and Mix6NR). For demonstration purposes and to compute the TPR value, we assumed a fixed decision threshold of $Th = 0.001$, which defines a strong sensitivity point in the validation set ROC curve. Figure 7 shows the ROC curves for the Mix6NR filtered subset (for both AE methods and a random baseline classifier), with the LSTM-AE method slightly outperforming the dense AE result.

TABLE IV: LSTM-AE subset test results for each anomaly use case (best results per mixture in **bold**).

| Mixture | Use Case | AUC | pAUC | TPR |
|---|---|---|---|---|
| Mix3 | Arguing | **92.67** | **83.28** | **92.06** |
| | Breaking Window | 72.58 | 57.59 | 49,21 |
| | Cough | 84.84 | 71.12 | 80,62 |
| Mix6 | Arguing | **89.89** | **77.59** | **86.61** |
| | Breaking Window | 71.15 | 53.01 | 54.44 |
| | Cough | 85.18 | 71.85 | 69.14 |
| Mix6NR | Arguing | **94.19** | **87.58** | **91.82** |
| | Breaking Window | 77.61 | 70.35 | 63.33 |
| | Cough | 88.07 | 78.59 | 83.85 |

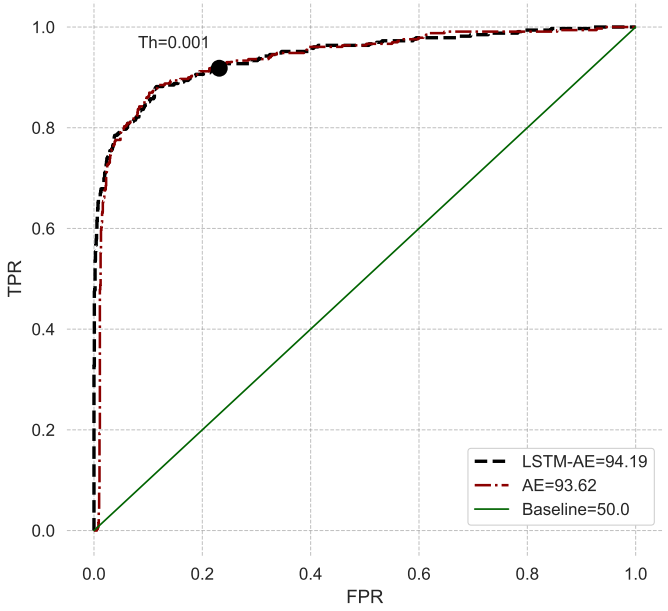


Fig. 7: ROC curves for AEs and random baseline classifier.

## IV. Conclusions

The advent of self-driving cars is expected to launch new business opportunities, such as autonomous shared taxicabs. In the absence of a company car driver, it is highly relevant to monitor the in-vehicle environment for safety, health and comfort issues. Set within a larger R&D project, this paper approaches an unsupervised in-vehicle Acoustic Anomaly Detection (AAD) system. Since public datasets are not available, we first designed a novel synthetic audio anomalous event simulator for in-vehicle AAD, which was used to generate three audio mixtures that include background trip sounds merged with five normal (e.g., people talking) and three anomaly events (e.g., cough). Then, we explored two sound feature extraction methods and a proposed Long Short-Term Memory Autoencoder (LSTM-AE) method to perform AAD. The method was compared to two state-of-the-art ADD methods: a dense Autoencoder (AE) and a two-stage clustering method (IRESE).

Overall, competitive results were achieved by the LSTM-AE architecture, which required an affordable computational effort and delivered interesting class discrimination results. In particular, a very good/good discrimination was obtained for the "Breaking Window" anomaly, while an excellent classification quality was achieved when detecting the other anomaly events ("Arguing" and "Cough"). In future work, we plan to extend the anomalous event simulator capabilities, for instance by considering Generative Adversarial Networks (GANs) to synthetically generate more sound events. Moreover, we intend to collect more real data by using human actors to mimic the abnormal events (e.g., fake realistically a cough). Another research direction is the adaptation of the proposed method to work with other types of data, such as provided by particle sensors (e.g., measuring air quality levels).

## ACKNOWLEDGMENTS

This work is supported by the European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project n 039334; Funding Reference: POCI-01-0247-FEDER-039334].

## REFERENCES

[1] M. M. Rahman, S. Deb, L. Strawderman, B. Smith, and R. Burch, "Evaluation of transportation alternatives for aging population in the era of self-driving vehicles," *IATSS Research*, vol. 44, no. 1, pp. 30 – 35, 2020.

[2] L. Sieber, C. Ruch, S. Hrl, K. Axhausen, and E. Frazzoli, "Improved public transportation in rural areas with self-driving cars: A study on the operation of swiss train lines," *Transportation Research Part A: Policy and Practice*, vol. 134, pp. 35 – 51, 2020.

[3] S. Kim, J. J. E. Chang, H. H. Park, S. U. Song, C. B. Cha, J. W. Kim, and N. Kang, "Autonomous taxi service design and user experience," *Int. J. Hum. Comput. Interact.*, vol. 36, no. 5, pp. 429–448, 2020.

[4] K. Drossos, S. I. Mimilakis, S. Gharib, Y. Li, and T. Virtanen, "Sound event detection with depthwise separable and dilated convolutions," in *IJCNN*. IEEE, 2020, pp. 1–7.

[5] S. Adavanne, A. Politis, J. Nikunen, and T. Virtanen, "Sound event localization and detection of overlapping sources using convolutional recurrent neural networks," *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 1, pp. 34–48, 2018.

[6] E. Cakır, G. Parascandolo, T. Heittola, H. Huttunen, and T. Virtanen, "Convolutional recurrent neural networks for polyphonic sound event detection," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, no. 6, pp. 1291–1303, 2017.

[7] Z. H. Janjua, M. Vecchio, M. Antonini, and F. Antonelli, "IRESE: an intelligent rare-event detection system using unsupervised learning on the iot edge," *Eng. Appl. Artif. Intell.*, vol. 84, pp. 41–50, 2019.

[8] T. B. Duman, B. Bayram, and G. Ince, "Acoustic anomaly detection using convolutional autoencoders in industrial processes," in *SOCO*, ser. Advances in Intelligent Systems and Computing, vol. 950. Springer, 2019, pp. 432–442.

[9] F. Aurino, M. Folla, F. Gargiulo, V. Moscato, A. Picariello, and C. Sansone, "One-class SVM based approach for detecting anomalous audio events," in *INCoS*. IEEE, 2014, pp. 145–151.

[10] S. Rovetta, Z. Mnasri, and F. Masulli, "Detection of hazardous road events from audio streams: An ensemble outlier detection approach," in *EAIS*. IEEE, 2020, pp. 1–6.

[11] P. Harár, Z. Galaz, J. B. A. Hernández, J. Mekyska, R. Burget, and Z. Smékal, "Towards robust voice pathology detection," *Neural Comput. Appl.*, vol. 32, no. 20, pp. 15 747–15 757, 2020.

[12] A. Farzad and T. A. Gulliver, "Unsupervised log message anomaly detection," *ICT Express*, vol. 6, no. 3, pp. 229–237, 2020.

[13] D. Kohlsdorf, D. Herzing, and T. Starner, "An auto encoder for audio dolphin communication," in *IJCNN*. IEEE, 2020, pp. 1–7.

[14] D. Y. Oh and I. D. Yun, "Residual error based anomaly detection using auto-encoder in SMD machine sound," *Sensors*, vol. 18, no. 5, p. 1308, 2018.

[15] Y. Koizumi, Y. Kawaguchi, K. Imoto, T. Nakamura, Y. Nikaido, R. Tanabe, H. Purohit, K. Suefusa, T. Endo, M. Yasuda, and N. Harada, "Description and discussion on DCASE2020 challenge task2: Unsupervised anomalous sound detection for machine condition monitoring," in *arXiv e-prints: 2006.05822*, June 2020, pp. 1–4. [Online]. Available: https://arxiv.org/abs/2006.05822

[16] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning," *Pattern Recognit.*, vol. 58, pp. 121–134, 2016.

[17] S. Bu and S. Cho, "Classifying in-vehicle noise from multi-channel sound spectrum by deep beamforming networks," in *BigData*. IEEE, 2019, pp. 3545–3552.

[18] X. Zhang, Y. Chen, and G. Tang, "Audio-based classification of automobile driving conditions," in *Proceedings of the 2019 International Conference on Artificial Intelligence and Computer Science*, ser. AICS 2019. Association for Computing Machinery, 2019, p. 808811. [Online]. Available: https://doi.org/10.1145/3349341.3349517

[19] P. Paulraj, A. M. Andrew, and Y. Sazali, "Car cabin interior noise classification using temporal composite features and probabilistic neural network model," *Applied Mechanics and Materials*, vol. 471, pp. 64–68, 12 2013.

[20] H. V. Koops and F. Franchetti, "An ensemble technique for estimating vehicle speed and gear position from acoustic data," in *DSP*. IEEE, 2015, pp. 422–426.

[21] P. Laffitte, Y. Wang, D. Sodoyer, and L. Girin, "Assessing the performances of different neural network architectures for the detection of screams and shouts in public transportation," *Expert Syst. Appl.*, vol. 117, pp. 29–41, 2019.

[22] A. Mesaros, T. Heittola, and T. Virtanen, "TUT database for acoustic scene classification and sound event detection," in *24th European Signal Processing Conference, EUSIPCO 2016, Budapest, Hungary, August 29 - September 2, 2016*. IEEE, 2016, pp. 1128–1132. [Online]. Available: https://doi.org/10.1109/EUSIPCO.2016.7760424

[23] B. McFee, C. Raffel, D. Liang, D. P. Ellis, M. McVicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proceedings of the 14th python in science conference*, vol. 8, 2015, pp. 18–25.

[24] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[25] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, pp. 861–874, 2006.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011. [Online]. Available: http://dl.acm.org/citation.cfm?id=2078195

[27] P. J. Pereira, P. Cortez, and R. Mendes, "Multi-objective grammatical evolution of decision trees for mobile marketing user conversion prediction," *Expert Systems with Applications*, vol. 168, p. 114287, 2021.