SCTE·ISBE CABLE-TEC EXPO 2019 NEW ORLEANS, LA SEPT. 30-OCT. 3

2019 Fall Technical Forum SCTE·ISBE • NCTA • CABLELABS

Smart Home & Smart Notifications

Multimodal event detection and notification

A Technical Paper prepared for SCTE•ISBE by

Henk Heijnen
Project leader
Technicolor France
975 Av des Champs Blancs CS17616 – 35576 Cesson-Sévigné - FRANCE
+33 2 99 27 3000
henk.heijnen@technicolor.com

Philippe Gilberton
Senior scientist
Technicolor France
philippe.gilberton@technicolor.com

Jean-Ronan Vigouroux
Senior scientist
Technicolor France
jean-ronan.vigouroux@technicolor.com

Table of Contents

Title **Page Number**

List of Figures

List of Tables

1. Introduction

In this paper we will explain the outcomes of our research project on the detection of anomalies in the home.

The goal of our experimentation is to be able to detect such anomalies as unexpected sounds (dog barking, gun shot, baby cry …) and reports from simple sensors measuring temperature or humidity for example.

1.1. Exceptional sounds recognition

Because listening to in-home sounds 24/365 is very critical regarding privacy, all the processing is done locally, and only recognized events are sent to the cloud / backend.

The technology we are using (neural networks) requires very large sound databases during the model build (10000+ sounds per class), therefore, the models are pre-calculated offline and are downloaded to the recognition brick. We can attain extremely good recognition rates although we can't learn new sounds nor improve the model locally.

1.2. Multimodal sensors, abnormal situations detection

Another strong constraint is that we do not want to force users to create scenarios for controlling the device behavior regarding sensors management. We have developed a self-learning, non-supervised technology that will allow our product to learn from "know good" situations and to report events that are outside of this zone without the user having to configure the product.

2. Exceptional Sound Detection

2.1. Introduction

This section aims to provide some insight into the tremendous gap that Neural Network (NN) architecture has delivered, over the last several years, in the sound classification space. This paper will also include a more detailed focus on current NN architectures, and their performances compared to one of the most popular classifiers relying on the Support Vector Machine (SVM).

2.2. Some definitions

First, let's provide the reader with some definitions that should be considered as well as the device requirements of a sound detector.

2.2.1. Rare/exceptional sound events

A rare/exceptional sound event is defined as a sound event that occurs once within a fixed period of time which is significantly longer than the event duration itself. A typical example is a glass breaking that would last less than a few seconds and would only occur at most once every couple of months. In the acoustic Machine Learning (ML) domain it signifies that there is a significant unbalance ratio between positive (occurrence of sound) and negative (no occurrence of sound) samples. Some years ago, the ML algorithms were trained on almost continuous and well-balanced audio samples which give good results with controlled test configurations but not so good in a real situation. This imbalance situation constitutes a challenging but more realistic problem to solve for a Machine Learning which must learn on few positive samples. Furthermore, it involves the usage of a supervised leaning approach to build a satisfactory acoustical model. For comparison purposes an unsupervised learning approach would need regular and recurrent positive samples which are not compliant with exceptional sound detection requirements.

2.2.2. Sound detector

A sound detector is a device whose purpose is to detect a sound event within a fixed temporal but sliding window. As shown in the upper rare/exceptional definition, the sound detection has a coarse sound detection granularity which means that the exact time of the sound event occurrence presented figure 1 (onset and offset time) is not provided. Additionally, the sound detector will operate in a domestic environment in which background noise may exist. As will be detailed further in this paper, this will have some consequences on both the training performance of the ML and the audio dataset composition.

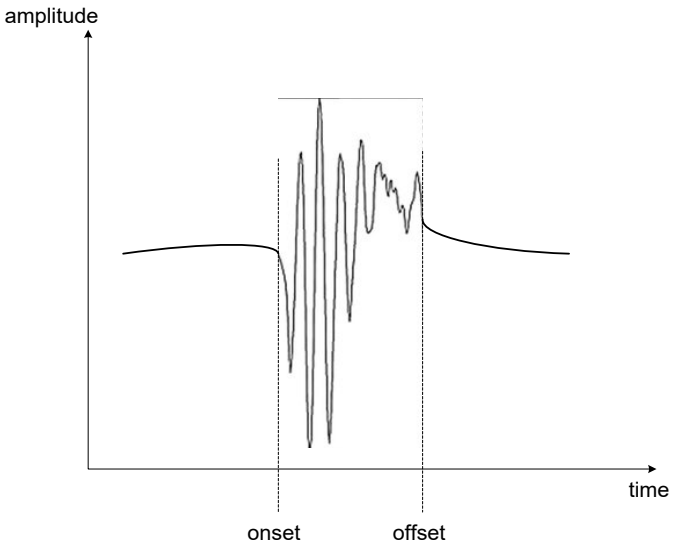

Figure 1 acoustic onset, offset time

This means that the ML must learn within an ambient noise environment and with a weakly annotated dataset. More specifically, a weak annotation means that, for each sample of sound used for training, the ML will only know that the sound of each class to recognize is present but without any indication of when it occurs within this audio sample. That is an important parameter to pay attention to as it makes easier to build the audio dataset, while another more complex approach, involving a detailed temporal annotation, would be too time consuming and not cost effective at the end.

2.3. ML based sound detection demystified

2.3.1. Sound detector service architecture

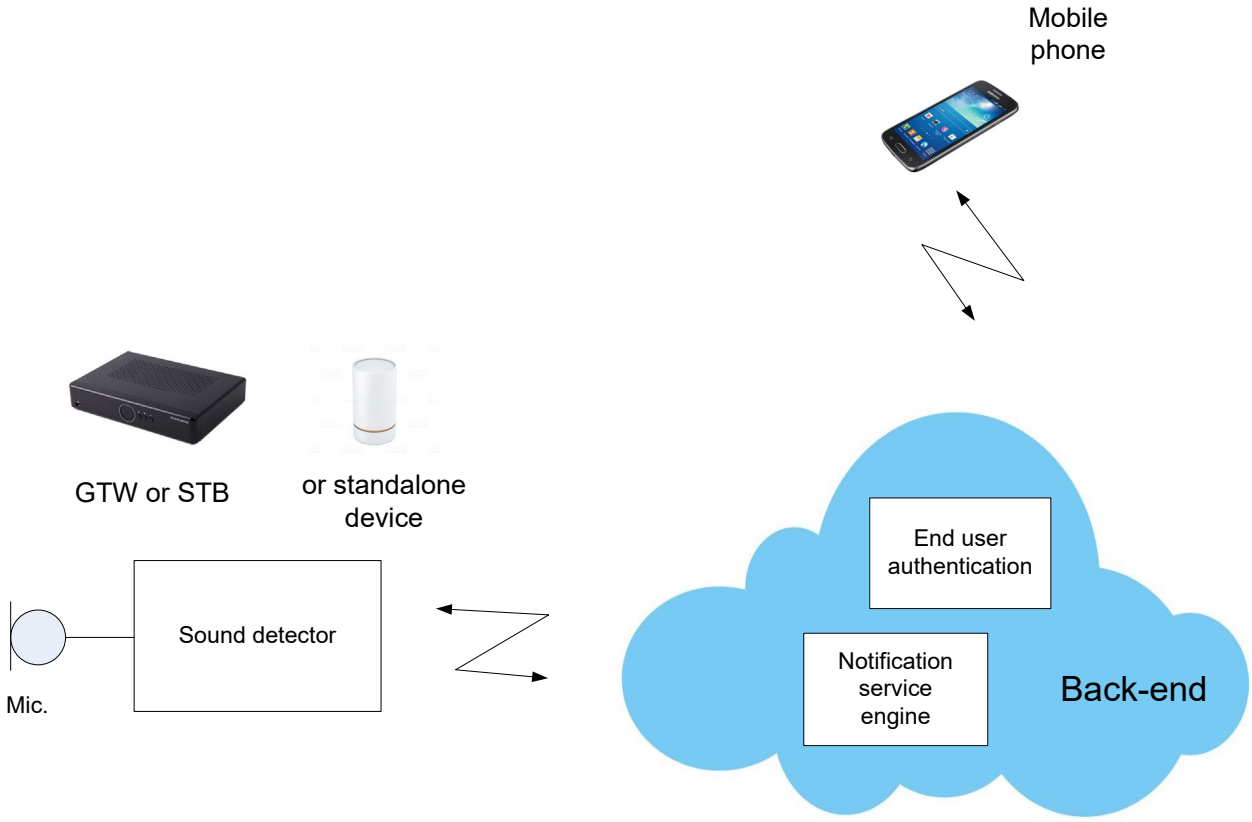

Figure 2 Sound Detector Service Architecture

This figure presents an example of a sound detection service architecture that would rely on a mono modal service approach (i.e. only the sound modality) but that could be easily integrated into a multi-modal approach, as exposed in the anomaly detection section (see section 3). In this simplified figure 2, the sound detector would provide the sound class label for each processed temporal window (i.e.: every 6 seconds) without sending and saving any audio samples captured in the household. The sound detector has been optimized to be embedded in small footprint hardware to preserve data privacy of the user (refer to §5). Such a local processing approach has a significant advantage versus cloud-based audio recognition systems like Alexa Guard because it protects user's privacy: the backend, with no access to the user data, will manage only the transmission of the notification to the user and/or the service provider and will not have access to audio samples. A smart notification system could be deployed according to the nature of the detected sound.

2.3.2. Sound detector functional architecture

As an introduction to a more detailed description of the sound detection system based on ML, an overview of a functional architecture of the system is presented in the figure hereafter:

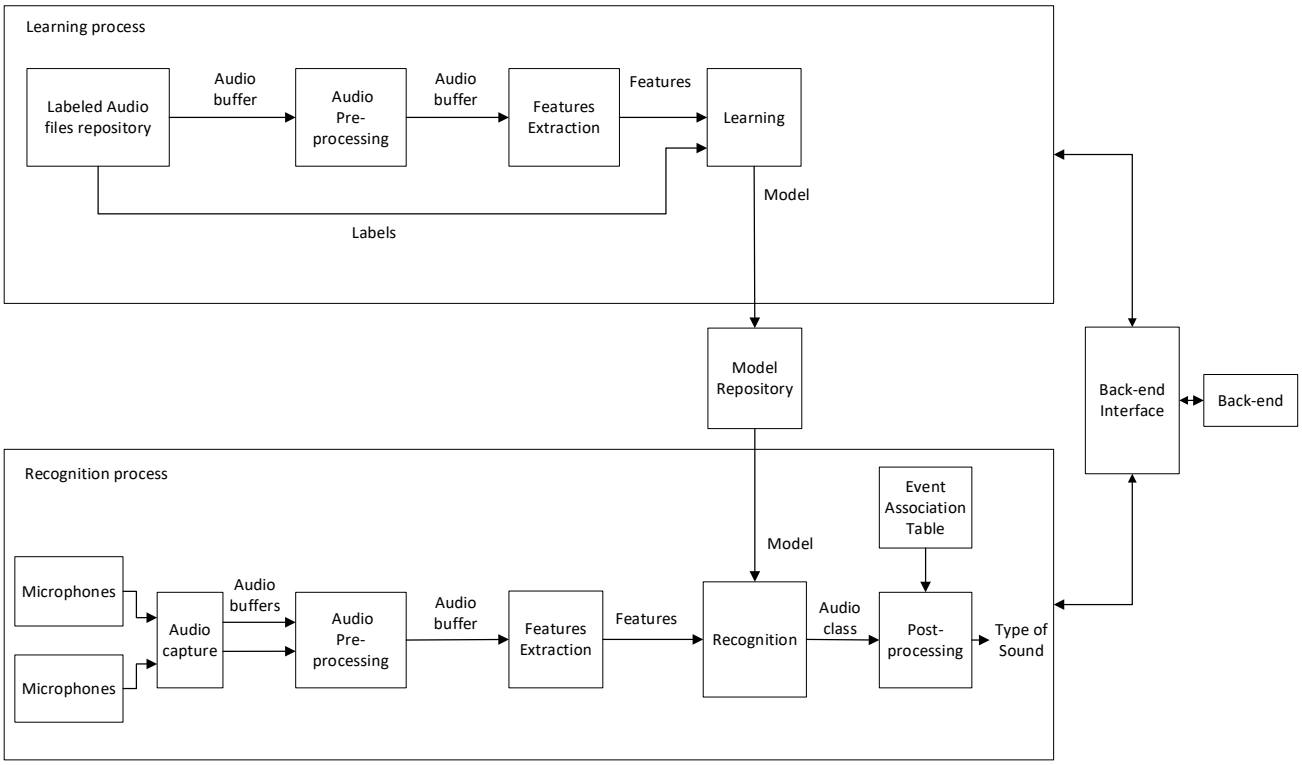

Figure 3 Functional Architecture

What can be noticed in figure3 is an architecture based on two blocks that operate two separate and sequential processing sessions. As in any ML system, those two required processes are first the training part, usually performed off-line and once to build the acoustic multiclass model, and then the acoustical detection part. As mentioned in paragraph 2.2 the ML approach is based on supervised learning which means that the acoustic model is trained on annotated dataset audio samples. This approach will justify building a huge dataset that is as diversified as possible (see paragraph 2.3.2.1).

Let's focus now on the sound detector functional architecture presented in Figure 4 Sound Detector Functional Architecture and a more detailed description of the design:

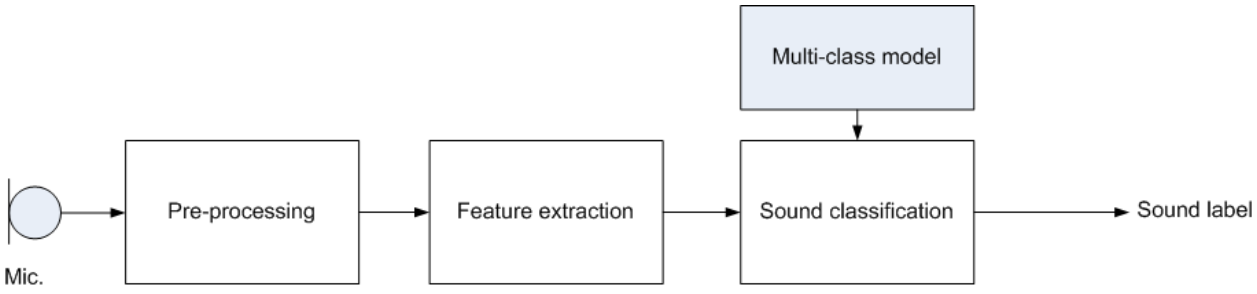

Figure 4 Sound Detector Functional Architecture

2.3.2.1. Dataset

Prior to training the supervised ML, getting the multiclass model and then evaluating its performance, a sound dataset shall be carefully built by following the typical folder organization:

A folder for the training set and a folder for the test set. Each of those 2 folders is subdivided into a number of folders according to the number of sounds to detect. Usually the file number ratio between training and test sets are 80% and 20% respectively as the learning effort must be put on the training session to get the best acoustical model as possible.

Each sound of the dataset is a mix of the sound event occurrences to detect and different background noises that fit with a realistic acoustical household environment. Many data sources are freely available like https://freesound.org/ or https://research.google.com/audioset/ for which a very convenient search engine tool is available. But even with such huge sound libraries, the acoustical diversity and sometime quality criteria are not reached, especially with sound classes that are not as popular (i.e.: hand clapping, snapping or coin rolling on a table). More recently, a new technique has emerged that uses data augmentation to dramatically diversify the dataset by adding in random manner small signal perturbations on the initial training set. Simple algorithms perform such synthetic data augmentations by adding noise, stretching time, changing pitch or speed or even adding acoustical room reverberation. This enriched training set is now much more suited to reflect the acoustic sound diversity present in different household environments.

2.3.2.2. Audio pre-processing

This function aims to capture audio samples from the microphone or the array of microphones with fixed parameters such as sampling rate, number of channels, compression algorithm if required and bit depth (i.e: 44.1 kHz, mono, uncompressed, 16bits). Optionally, frequency filtering is applied to the audio signal to minimize the effect of sensitivity dispersion and non-linearity of the microphone. A rms input power level estimation may also be performed to disable the classification process when the input power audio signal is under a fixed threshold value.

2.3.2.3. Features extraction

This function is essential in ML usage as it will extract the relevant acoustic characteristics that will be used to train the ML. There are different techniques available, the most popular of which is the computation of MFCC [1] (Mel Frequency Cepstrum Coefficient) presented hereafter:

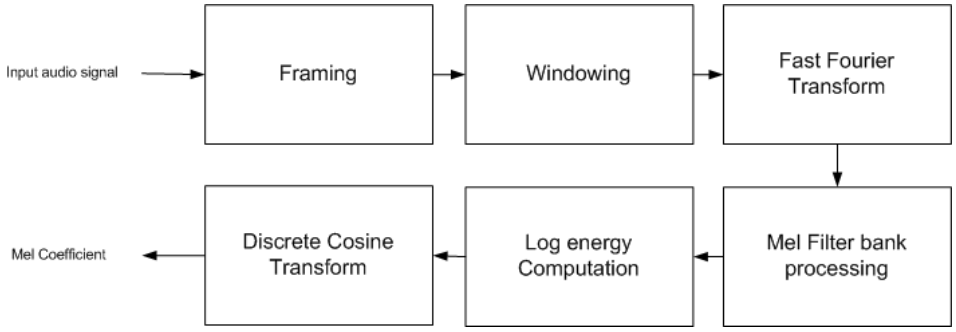

Figure 5 MFCC Computation Pipeline

Prior to computing the features, framing and windowing functions are required. This is based on applying a sliding temporal window that divides the incoming audio samples into small frames (ranging usually from 10 to 50 ms). The frame duration determines the temporal resolution targeted by the system. To avoid acoustical artefact on the edge of the frames, a Hann or Hamming algorithm is applied with an overlapping ratio of 50%. The signal is then ready for FFT processing. As the signal is now transposed in the frequency domain, a filter bank is applied with a Mel frequency spacing scale. The number of digital filters constituting the filter bank fixes the number of Mel coefficients that are computed. The Mel coefficients

are followed by a normalization function that usually uses a logarithm function, but more recently, the normalization of Mel coefficients has been improved by using PCEN [2] (Per channel Energy Normalization) that have a better signal saliency property in the presence of background noise. The number of coefficients is a parameter to fix to a value usually ranging from 10 to 100 depending on the targeted spectral frequency resolution.

2.3.2.4. Audio classification

This function is critical as it performs the probabilistic classification of sounds per captured audio samples. In other words, it will provide a probability value per audio class for which the sum must equal to one. The detected sound will be the one which gets the highest probability value. A threshold can be applied at the output based on the maximum of the computed probabilities. From a satisfactory user experience prospective, the threshold must be determined carefully to get a fair balance between an acceptable low level of false positive (wrong classification of a sound event occurrence) and an acceptable low level of false negative (missing of a sound event occurrence).

To provide more focused insight on the state-of-the-art classification algorithms, only SVM (Support Vector Machine), the most popular one, and NN (Neural Network) approaches will be presented.

2.3.2.4.1. SVM based algorithm classification

The SVM (Support Vector Machine) is part of a family of algorithms well suited to classification, regression or anomaly detection problems. They were initially developed in the 90s and rely on finding the simplest way to separate the classes of data by maximizing their distance. The edge of separation of those data is also called the margin. For audio purpose classification problems, the classes of data correspond to the different type of sound to recognize. The acoustical data are represented by vectors which are the extracted features computed in §2.3.2.3. In a simple binary classification problem, the vectors located on the edge of the separation of the 2 classes are named the support vectors as illustrated in figure 6:

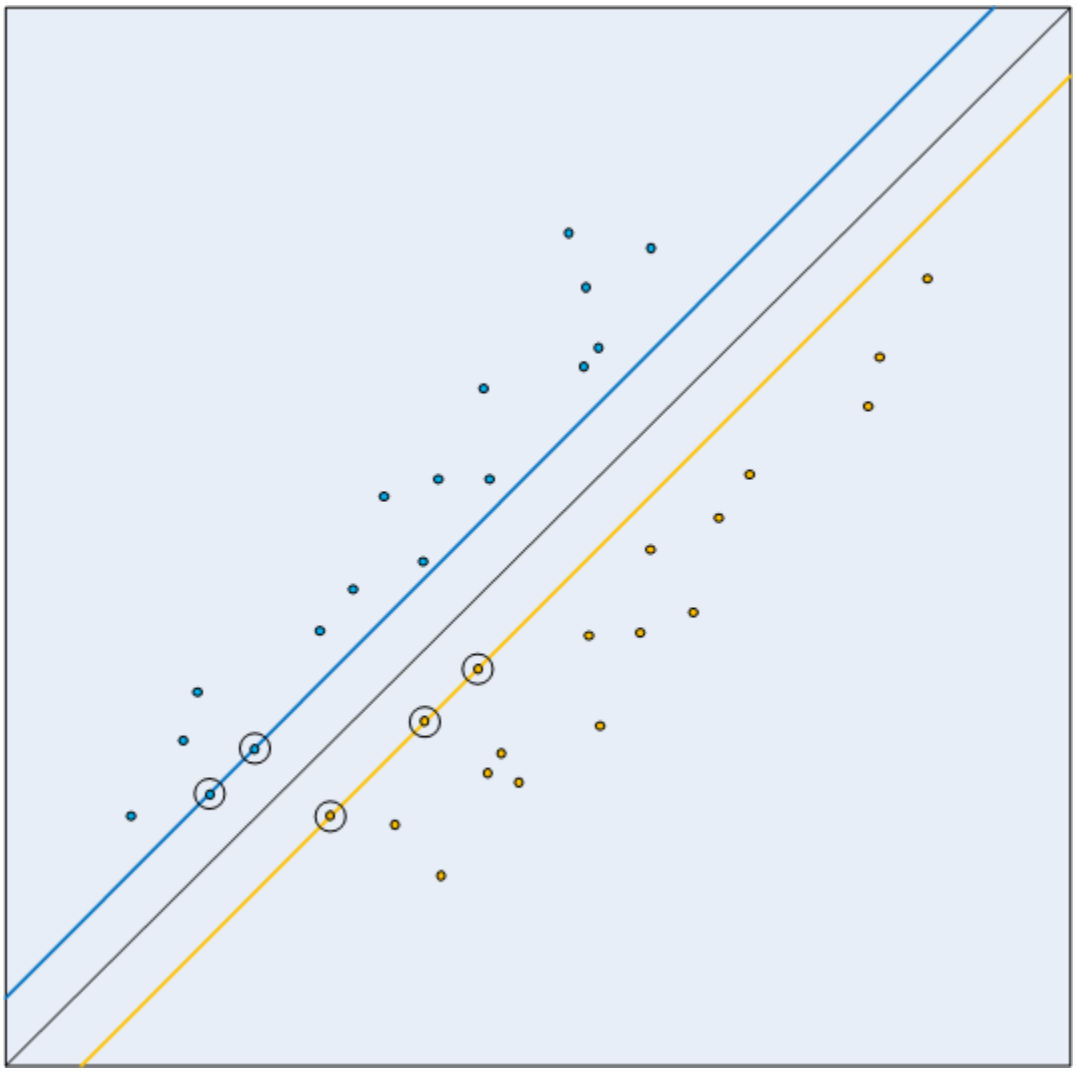

Figure 6 SVM Vector View In The Case Of A Binary Classification

In this simple example, the data are represented in a 2-dimensional space, where the edge of separation of data is the black line and the margin is the range between the black line and 2 blue and yellow lines. The support vectors that are closest to the edge are represented by circled blue and yellow dots. In this particular case, a linear separation is possible but in most actual cases this is not true, and other techniques are required. They are named kernels and allowed to separate data by projecting them in a higher dimensional space. Those kernels are based on polynomial or gaussian functions. SVM is convenient to setup as it requires few hyperparameters that are usually the regularization function, the kernel function and C which is a coefficient that penalizes more or less the cost of misclassification.

2.3.2.4.2. NN based algorithm classification

As an introduction, we will present the NN principle. Shown hereafter is a typical NN architecture:

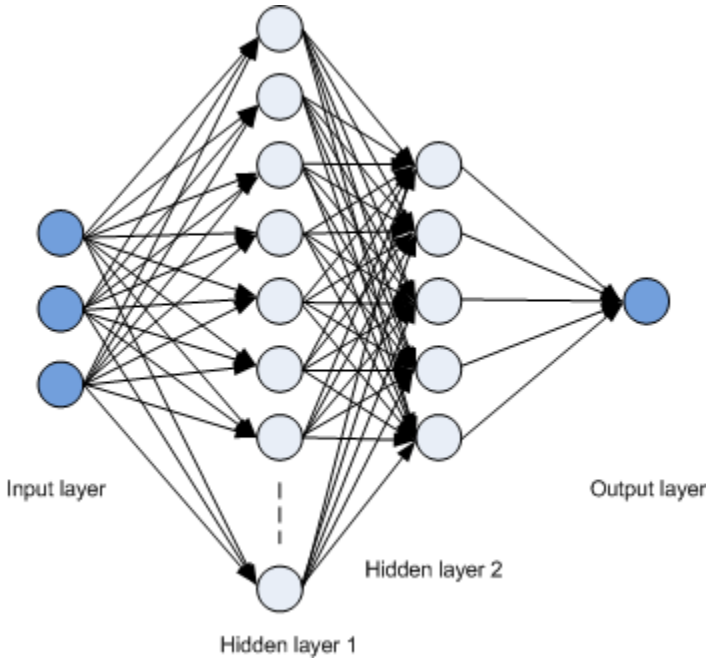

Figure 7 Typical NN Architecture With 2 Hidden Layers

The circle represents the neuron for which a weight and a bias are applied, and the arrow represents the link between one neuron to the next one, also going from the previous layer to the next one. In ML, a neuron makes a linear combination of input data (i.e.: extracted feature vectors) on which a bias value is added. The resulting output data value goes through a non-linear activation function (i.e.: hyperbolic tangent or sigmoid). This is performed for all neurons of the first NN layer (input layer). Then those output values are transmitted to the next NN layers (hidden layers) which will perform the same data computation and, finally, will end with an output layer that will provide an example of a classification problem, allowing the probability of each label to correspond with the audio classes that are necessary to be recognized.

What is remarkable is the simplicity of the elementary computation functions that are a combination of addition and multiplication. However, their number could be large as they increase from one layer to the next, except for the last layers. Compared to SVM, NN requires the selection of many hyperparameters to finally get to the most accurate multiclass model. The most common of those are typically the number of hidden layers, the type of layers (convolutional, recurrent, fully connected, …), the number of neurons per layer, the activation function, the number of epochs, the optimization function. In addition to that some other intermediate functions are added like batch normalization, dropout or pooling for which specific parameters must be selected as well.

2.3.2.4.2.1. Full trained NN model

Full trained NN model means that the model is trained and tested with the entire available dataset (see §2.3.2.1). The NN architecture used for training relies on CNN for which a simplified architecture is presented below followed by several acronym definitions:

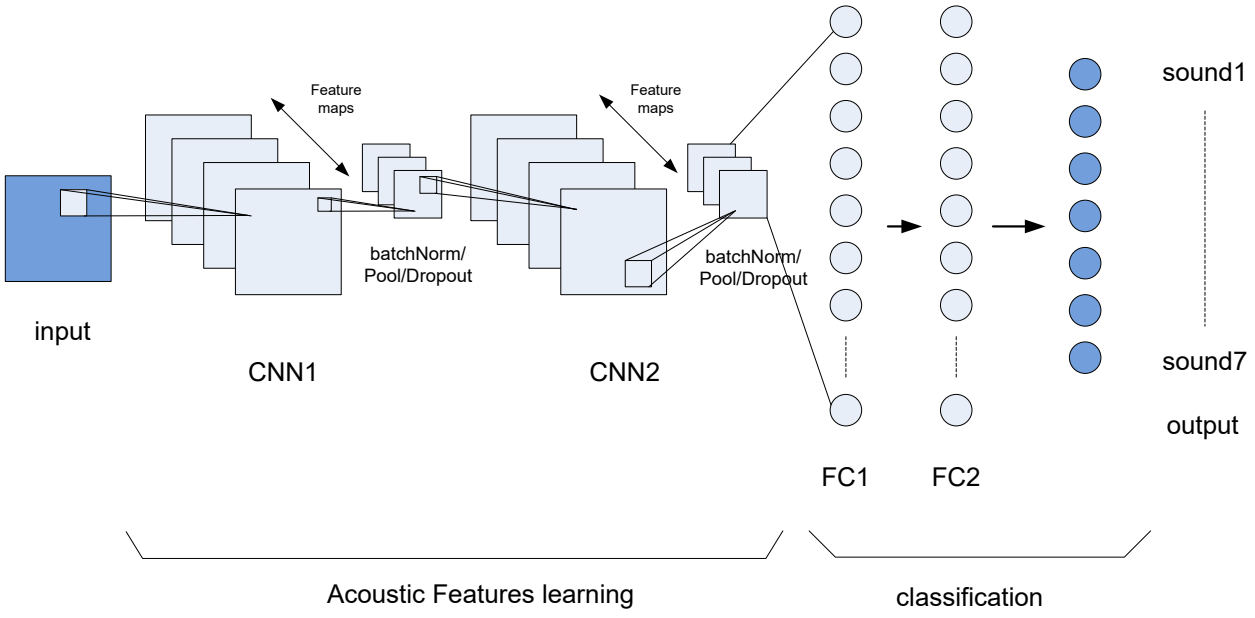

Figure 8 Exemplary Of A Two CNN Layers Architecture

The input is a 2-dimensional acoustic features matrix with columns and rows respectively corresponding to the number of acoustic features coefficients and frames. Then the NN architecture is split into 2 parts, the feature learning one including the CNN layers and the classification one including the fully connected layers that ultimately provide the probability of each sound class required to be recognized.

Feature maps are the result of convolutional operations using a small filter matrix performed at each location of the CNN layer. The number of filters determines the level of granularity you would like the NN to be trained on.

BatchNormalization is a normalization function performed on the incoming features to adjust and scale the activations. If helps learning in stabilizing the NN weight computation especially when the dataset is diversified and for which the feature values vary a lot from 1 class to the other.

Pooling (also named MaxPooling) is a downsampling function to reduce feature dimension and minimize overfitting. Usually the maximum value of a subpart of the filter matrix is computed which is the reason the name usually has the prefix of Max.

Dropout is a regularization function that randomly deactivates some neurons. The effect is that the NN becomes less sensitive to the specific weight of neurons. As a consequence, the NN model is capable of better generalization and is less likely to overfit on training data.

FC (Fully Connected) layer also named Dense layer executes a linear operation on the input vector and is located at the end of the architecture to transition from CNN layers. In a multiclass classification problem, for which a probability per class is required, a softmax function is applied in the last Fully Connected layer.

Another type of NN layer, less popular for sound classification because of the occurrence of model convergence problems and of a much longer training time, is the RNN (Recurrent Neural Network). The main difference from the CNN approach is that the next vector value depends not on the previous one but also on all the previous history. RNN has a memory of what happened in the past compared to CNN. The

last generation of RNN uses LSTM (Long Short-Term Memory) that improves during the training phase model convergence problem like gradient vanishing effect. Some results will be exposed in § 2.3.2.4.3.

2.3.2.4.2.2. Pre-trained NN model

The pre-trained NN model has become more popular meaning the model is already trained on a large dataset and is by consequence very well generalized for most of the classification/segmentation/object detection problems. Many pre-trained models exist for image processing (Xception, VGG19, ResNet50, InceptionV3, MobileNet, etc…) and fortunately some are available for sound classification as well like:

- VGGish(https://github.com/tensorflow/models/tree/master/research/audioset). It uses CNN architecture and it takes as a spectrogram an input dimension of 96 x 64 followed by 4 CNN and maxPooling layers. It ends with a 128-wide fully connected layer ready to be linked for example to a last FC layer for sound classification purpose.
- soundNet(https://github.com/eborboihuc/SoundNet-tensorflow). Its architecture is presented below:

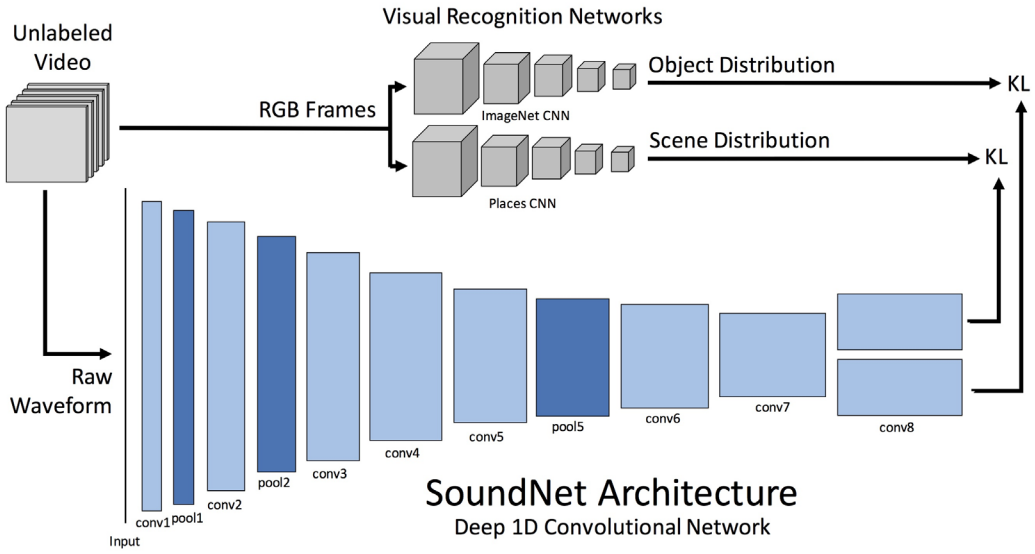

Figure 9 SoundNet NN architecture

SoundNet [3] is composed of up to 8 CNN layers and 5 maxPooling layers. As shown on figure 9, SoundNet addressed the acoustic scene/object classification and significantly improved the state-of-the-art results.

The main advantage of using this technique is to benefit from the quality of the model that is well generalized because it was trained on a large dataset with many NN layers. Typically, VGGish used the audioset (https://research.google.com/audioset/) database which contains more than 2 million annotated videos representing around 6000 hours of sounds. What constitutes the main constraint is you must strictly follow the same input data format that was used for training the model.

Those pre-trained models could be used in 2 ways: first, as a feature extractor or second, to fine tune the pre-trained model to better fit it to your own dataset. In the first case it means you don't care anymore about how to extract features because the pretrained model, based on NN, will do it for you. The other benefit is the significant reduction of training time as only the last fully connected layers are required (with optionally the dropout function) because the NN layers were already used in the pre-trained model. In the second case,

there is not that much training time saved as the pre-trained weights of the model not only have to be updated on all or a part of the NN layers but also on your own dataset.

An interesting comparison on performance of classifiers, based on the usage or not of a pre-trained model, is summarized in this paper [4]. In this case, when using it as a feature extractor in a binary acoustic classification problem, it clearly demonstrates the accuracy improvement:

Table 1 : Classification results on the training, validation and test set using different audio feature extractors

Audio features	#dim.	Accuracy %		
		Training	Validation	Testing
MFCC	1212	83.10	83.84	84.05
MFCC	128	89.84	84.93	84.79
CQCC	1404	87.58	84.64	84.95
CQCC	128	86.59	85.53	85.29
SoundNet	512	89.45	86.00	86.87
VGGish	128	88.58	86.05	88.04
VGGish+SoundNet	**640**	**90.40**	**88.54**	**90.43**
VGGish+SoundNet + CQCC+MFCC	512	89.77	86.36	88.68
Baseline model of ASVspoof 2017	2223	76.31	76.30	72.63

2.3.2.4.3. NN versus SVM performance comparison results

Another comparison was performed to justify the significant improvement of using NN versus SVM approaches. That was a performance evaluation conducted on 2018 for each classifier in using the same rare sound database extracted from the acoustic challenge DCASE2017 (http://www.cs.tut.fi/sgn/arg/dcase2017/challenge/task-rare-sound-event-detection).

The generated dataset consists of isolated sounds (Baby cry, Glass breaking and Gunshot) mixed with different types of background noise (Beach, Bus, Cafe/Restaurant, Car, City center, Forest path, Grocery store, Home, Library, Metro-station, Office, Park, Residential area, Train, Tram). The sound event occurred only once over the 30 second duration of incoming sound and located randomly within the file. The signal to background noise ratio was also adjustable and randomly chosen among 3 values, -6, 0 and +6dB. 500 files per class were generated both for training and test phases. The challenge relied on the low level of the event occurrence and the level of background noise. It meant the model had to be trained on a very unbalance training set with a lot of negative data (no event) or very little positive data (event). The results are summarized in the table 2 hereafter:

Table 2 : SVM versus NN classifier performance comparaison

classifier	Event F-score				Event Error Rate			
	average	babycry	glassbreak	gunshot	average	babycry	glassbreak	gunshot
RNN	**93.10%**	92.20%	97.60%	89.60%	**13.00%**	15.00%	5.00%	19.00%
CRNN	**85.12%**	81.99%	97.03%	76.33%	**28.00%**	38.00%	6.00%	39.00%
SVM(kernel rbf)	**75.93%**	81.51%	86.67%	59.63%	**44.00%**	43.00%	27.00%	61.00%

CRNN is an architecture that combines CNN and RNN layers. The metrics used during the test phase are listed as follows:

- the event F-score ratio computes the ratio of successful event detections and positioning per audio test file
- the Event error rate computes the ratio of misclassification and bad positioning per audio test file

Table1 clearly shows that, in this particularly challenging but realistic audio dataset, the NN architecture outperforms the SVM one by at least 10% for the event F-score.

2.4. Conclusion

Exceptional sound detection, that a decade ago was a difficult ML challenge to overcome, has now demonstrated promising results thanks to the availability of NN architectures. The emerging usage of the pre-trained models inspired from the last tremendous advancements done on imaging computation (face recognition and tracking, object detection and tagging, etc...) is another track to explore by refining the detection of more complex sequences of sounds composed of multi labelled occurrences.

3. Anomaly Detection

3.1. Introduction to Anomaly Detection

3.1.1. Introduction

Machine Learning is an essential tool to implement Artificially Intelligent systems, that is systems analyzing and making decisions about their environment. The most well-known problems in Machine Learning are:

- Supervised Learning: learning to recognize classes of objects from examples,
- Unsupervised Learning: learning to find similarities between objects, learning to group or to represent objects without a class objective,
- Reinforcement Learning: learning to select the most profitable actions.

Aside from these points, Anomaly Detection is a lesser known subject that however has a huge interest for IOT devices. The idea is to learn to detect unusual or anomalous situations without examples of these situations. Put more clearly, the idea is to learn the normal behavior of a system and to trigger alarms when an unusual situation occurs.

This is particularly interesting for IOT as it is impossible for the end user to produce examples of anomalous situations (think of glass breaks or gunshots at home), or to label them. In addition, the variety of anomalous situations may be so huge (think of the different ways an old person may fall at home), that it looks much more reasonable to carefully model the normal situations and to trigger an alarm when the situation measured by the sensors deviates from normality.

It should be noted that the objective of these methods is to detect anomalous situations, and not to identify them. When an alarm is risen it will be necessary for the end user, or a third party (relative, neighbor, operator…) to check what happens, and see if there is really an urgency or if it is a false alarm, and to select the appropriate action.

3.1.2. An example of Anomaly Detection algorithm

Being able to detect anomalous situations without having seen any of them may seem puzzling. Let us however present an algorithm that will convince the reader that this is feasible. The objective of an anomaly detection algorithm is to compute a score that is expected to be low (or even negative) in a normal situation, and high (and generally positive) in an abnormal situation. The k-nearest neighbors can be used for anomaly detection as follows. Let us suppose that all the inputs to the algorithm are vectors in a n-dimensional space, for instance a set of n simultaneous measures. Let us suppose that we have a collection of N normal points at our disposal. For a new point one measures the distance to the k-nearest neighbor, that is the distance of the k closest point. This distance is clearly an interesting candidate for being an anomaly score, as a point close to many others is likely to be a normal point, whereas a point remote to all the other points is likely to be an anomalous point.

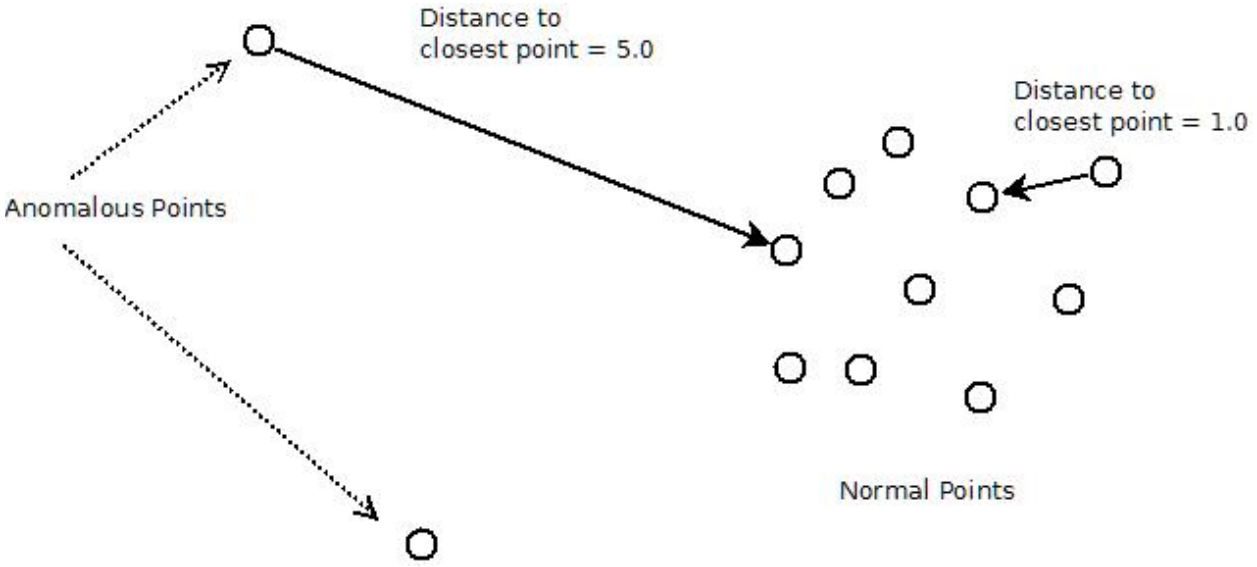

Figure 10 k-NN algorithm

In general, one will take k equal to a few units or tens, to avoid producing a low score for repeated anomalous points.

3.1.3. Different kinds of anomalies

The anomalies are unexpected events or situations occurring in the input stream. Following Chandola [5] we distinguish three kinds of anomalies.

- Point anomalies occur when an input vector is clearly different from all normal input vectors. This occurs for instance when a sensor is out of order and returns erroneous values, or an excessive temperature is measured, indicating a possible fire.
- Contextual anomalies correspond to situations which are abnormal in the context in which they occur. For instance, somebody preparing their breakfast in the middle of the night if this is not their habit.
- Collective anomalies occur when each measure in a segment of a series is normal, but the whole segment is anomalous. For instance, a fridge consuming no electricity for many hours.

It will appear that detecting point anomalies can be readily implemented, whereas the two other kinds of anomalies are more difficult to tackle.

3.2. Anomaly Detection Algorithms

Anomaly detection algorithms aim at computing a score that will characterize the likelihood that a situation is abnormal. It is expected that a low (or negative) score will be produced in normal situations, and a high (positive) score will be produced in anomalous situations.

Algorithms can be divided in two kinds. Many algorithms consider samples independently. However, in IOT situations, it is reasonable to hypothesize that there is a continuity between the successive measures of a sensor (for instance the temperature in a house is not likely to change very fast in normal situations). Therefore, it is interesting to use this continuity principle to characterize normal situations.

Below, we present examples of these two kinds of algorithms, after a reminder on algorithm performance evaluation.

3.2.1. Algorithms performance evaluation: the ROC curve

ROC curves are the classical way to evaluate detection algorithms, since the early days of radar technology.

A detection algorithm produces a score f (x) signaling the likelihood of a detection. If we fix a threshold S on the score this defines a classification function D (x):

$$D(x) = \begin{cases} True\ if\ f(x) > S \\ False\ if\ f\ (x) \leq S \end{cases}$$

Knowing a ground truth on the objects, it is possible to characterize the performance of the classification by the numbers of:

- True Positives (TP): objects which should be detected and are actually detected,
- True Negative (TN): objects which should not be detected and are actually not detected,
- False Positive (FP): objects which should not be detected and are actually detected,
- False Negative (FN): objects which should be detected and are actually not detected.

The performance of the classification function for a value S of the threshold is defined by the two ratios:

- True positive rate: $\text{tpr} = \frac{\text{TP}}{\text{TP} + \text{FN}}$
- False positive rate: $\text{fpr} = \frac{\text{FP}}{\text{FP} + \text{TN}}$

Letting the threshold S vary defines a curve named the Receiver Operational Curve, located in the square $[0, \ 1] \times [0, \ 1]$. The area under this curve AUC is the measure of the performance of the algorithm. It shows if the algorithm can have a high true detection rate with a low false positive rate.

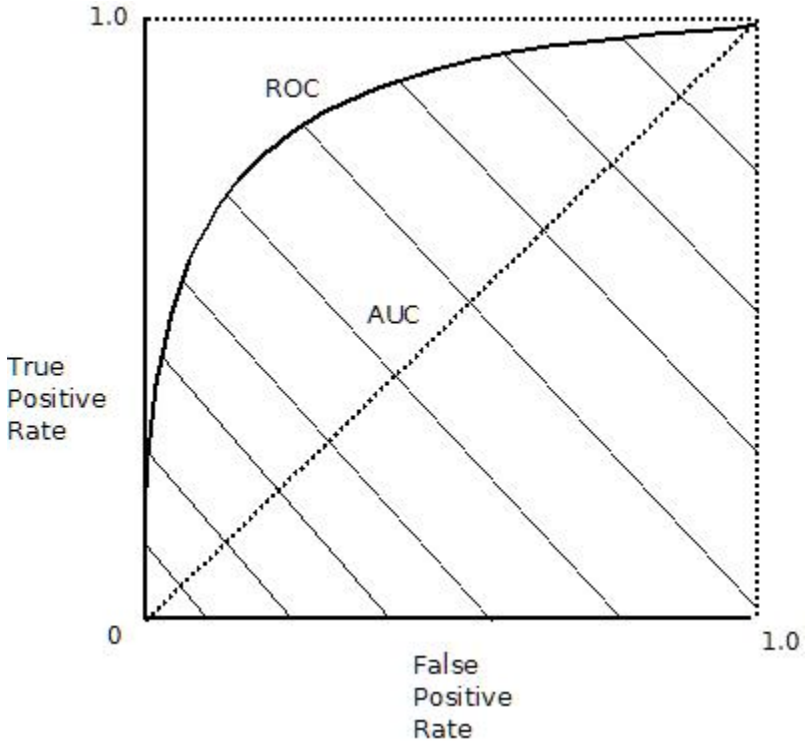

Figure 11 Receiver Operational Characteristic

The AUC is expected to be as close as possible to 1.0. If AUC is equal to 0.5 then the algorithm behaves as a random decision.

3.2.2. Algorithms not using the temporality

Many algorithms can be used to define an anomaly score, considering the successive measures as independent realizations.

Gaussian mixture models (GMM) model the series of samples as realizations of a mixture of Gaussians. Each sample is supposed to be produced by a two-step process: first select a Gaussian in a set of *n* Gaussians, each having a probability, and then draw a sample using this probability. The parameters of the model can be learned using the EM (Expected Maximization) algorithm [7].

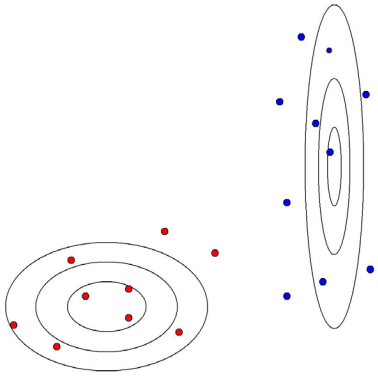

Figure 12 GMM Model

One-Class SVM is another example of classical algorithms aiming at detecting anomalies. The idea here is to separate the examples from the origin in a high dimension space, using a non-linear transformation.

Neural algorithms (see 2.3.2.4.2) have been used for detecting anomalies. A well-known example is the use of autoencoders. In this algorithm the parameters of the neural network are optimized to reconstruct the input vector on the output, after a projection in a lower dimension space in the middle of the network. In this case the anomaly score will naturally be the reconstruction error.

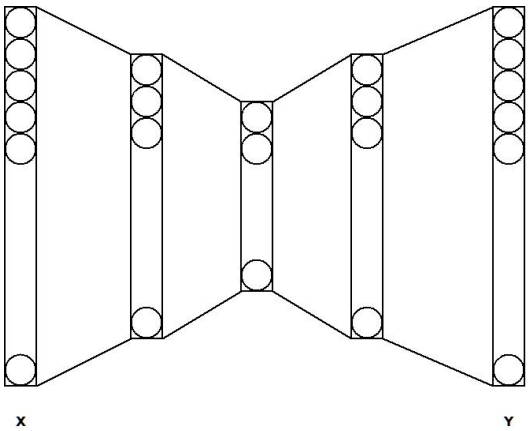

Figure 13 Autencoder

In an autoencoder the parameters of the network are optimized to make the input *X* of the network as close as possible to output Y, despite a projection on a lower dimensional space.

In general, these algorithms exploit the fact the input vector in normal situations does not take all the possible values of the sensors, but rather confines to specific regions of the input space, such as regions around specific points. In other words, this signal is compressible.

3.2.3. Algorithms using the temporality

In IOT situations the sensors tend to measure continuous data, such as temperature or pressure. The next temperature measure is in general very close to the previous one, and a deviation to this is generally a hint that something abnormal is happening. It is therefore pertinent to set-up a prediction algorithm that will try to estimate the next sensor measures, and to use to prediction error as the anomaly score.

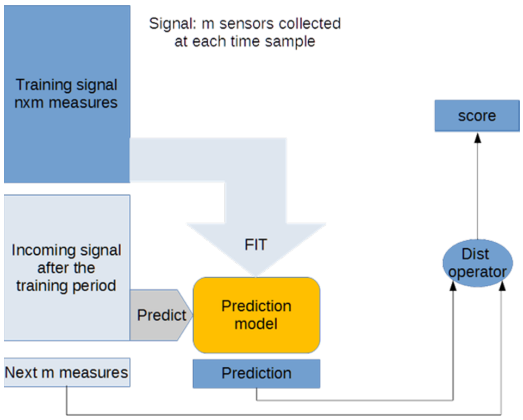

Figure 14 Training and Detection using Temporality

In this case a segment of the input signals is used to train a predictor. This predictor is used for a certain period, until the input signal has changed or after a fixed amount of time, when the predictor is re-trained. At each time step the already known examples are used to predict the next measures. The actual measure is compared to the prediction, and the deviation is the anomaly score. Many algorithms can be considered to implement the predictor.

Classic non-neural algorithms have been used for a long time to implement predictions.

The moving average is an example of such a simple algorithm. In this case the next sample is predicted by:

$$\widehat{x_i} = \frac{1}{N}\sum_{k=1}^{N} x_{i-k}$$

This simple algorithm does not imply any training phase. Similarly, the exponential smoothing uses the previous measures and the previous prediction to produce the estimation of the next measure.

$$\widehat{x_i} = \alpha\, x_{i-1} + (1-\alpha)\widehat{x_{i-1}}$$

In this case all the previous measures (potentially an infinite number) have an influence on the measure, with an exponentially decreasing weighting.

Holt-Winters algorithms are an example of multi-level regression, enabling it to consider seasonality effects in the data.

We have evaluated many neural architectures for providing the estimation of the next measure. The simplest architecture is a fully connected neural network, which takes the already known measures as input and produces the estimation of the next measure. See paragraph 2.3.2.4.2 for a description of dense neural networks.

Such a network can be implemented using a well-known library such as Tensorflow or Pytorch, trained using a retro-propagation algorithm.

Convolutional networks are an evolution of these networks, where weights are shared on the first layers to implement convolutional filters.

Finally, LSTM (Long Short-Term Memory) is a recurrent architecture, where the output of the network is reused to make the next prediction.

3.3. Evaluation

We have evaluated the algorithms on a dataset provided by EDF [6], the French electricity provider. This dataset is composed of measures of voltage, intensity and power in different part of a house, one measure per minute over a duration of four years. Each measure point is composed of seven individual measures.

We have introduced known anomalies on this dataset to measure the performance. The anomalies are:

- Out of order sensor: random values in the same min/max interval that the initial distribution,
- Activities at unexpected times: permutate some segments of the data,
- The fridge is stopped: subtract the fridge consumption from the meter in the related room.

We recognize here instances of point anomaly, contextual anomaly and collective anomalies.

Different algorithms were tested on this dataset and the anomalies specified above. The algorithms are not optimized to try to obtain the best results for each of them, but are rather parametrized to operate in comparable conditions:

- The temporal algorithms use a buffer of one-hour length (60 measures) to make the prediction,
- The algorithms are periodically retrained after 60x24x61 = 87840 measures, that is every two months,
- The dataset for training the algorithms (for the ones that need a training) is equal to at most eight months.
- The algorithms are trained on the normal data and the performance is measured on anomalous data. This is to avoid the problem of deciding what to do in the training with the data that is declared anomalous: skip it? Replace it by most probable values? This renders the evaluation of the algorithms far easier.

The conditions seem to be realistic for the data that we have used: we expect to operate the different algorithms on a small device, with limited memory and computation resources, as described in section 5.

This enables us to estimate what is the relative performance of the different algorithms in comparable conditions. It does not give the optimal performance that could be reached if the parameters of an algorithm were optimized using for instance a grid search method.

The algorithms are implemented using Python 3.7, Scikit-learn, Keras and Tensorflow. The detailed parameters of each algorithm are as follows:

- Moving average:
  - The prediction is the moving average of the series on the prediction window, that is one hour (60 elements).
- Exponential smoothing:
  - The alpha factor (see paragraph 0) is equal to 0.01
- GMM likelihood:
  - The number of kernels is equal to 16, the covariance for each kernel is a full matrix.
- Dense Neural Network:
  - The size of the input layer is always 420 (60 samples of size 7). The output layer is always of size 7. Different number of intermediate layers with 60 or 420 neurons have been tested. Different activation functions: *relu*, *tanh*, *sigmoid* have been tested.
  - The network noted *60 relu sigmoid* below corresponds to a network with 420 neurons on the input layer, then *relu* as activation function on an intermediate layer of 60 neurons, then. *sigmoid* as activation function, then the output layer which has size 7.
  - The network noted *420-60 tanh, tanh, tanh* is a network with 420 neurons on the input layer, a first hidden layer of 420 neurons, a second hidden layer of 60 neurons and an output layer of 7 neurons. The activation function between the input and the first hidden layer, the first hidden layer and the second hidden layer, the second hidden layer and the output layer are all *tanh*.
- Convolutional Neural Network:
  - The network is composed of an input layer with 420 neurons, a convolution layer Conv1D with 20 or 40 filters of length 8, a max pooling layer (pool size = 2), a second Conv1D layer with the same configuration as the first one, a max pooling layer (pool size = 2), and a dense layer with 7 neurons. The activation function is the *relu* function.
- LSTM:
  - The network is composed of two layers of LSTM functions with 60 units.

The AUCs measured for each algorithm are reported hereafter. We have retained only the best results for each algorithm to limit the data.

Table 3 : AUC of the scoring algorithms

Algorithm	**Anomaly 1: Point anomaly**	**Anomaly 2: Contextual anomaly**	**Anomaly 3: Collective anomaly**
Moving average	0.832	0.663	0.507
Exponential smoothing	0.817	0.629	0.505
GMM likelihood	**0.961**	0.496	0.503
Dense Neural Network 60 relu, sigmoid	0.936	0.603	0.518
Dense Neural Network 60 sigmoid, tanh	0.923	0.622	0.511
Dense Neural Network 420-60 relu, relu, relu	0.818	0.521	0.492
Dense Neural Network 420-60 tanh, tanh, tanh	**0.929**	0.595	0.508
Dense Neural Network 420-420-60 relu, relu, relu, relu	0.855	0.536	0.448
Dense Neural Network 420-420-60 tanh, tanh, tanh, tanh	0.866	0.568	0.507
Dense Neural Network 420-420-420-60 relu, relu, relu, relu, relu	0.850	0.529	0.469

Dense Neural Network 420-420-420-60 tanh, tanh, tanh, tanh, tanh	0.925	0.579	0.509
Convolutional Neural Network 20 filters, filter length = 8	0.900	0.647	0.513
Convolutional Neural Network 40 filters, filter length = 8	0.907	0.648	0516
LSTM nb_units = 60	**0.967**	0.569	0.529

3.4. Conclusion

3.4.1. Choice of the algorithm

The LSTM is the best algorithm (AUC = 0.967). The dense neural networks performance is lower (AUC = 0.929).

The GMM is nearly as good (AUC = 0.961), but its training time is far lower than the Neural Networks.

The numbers are not the optimal performance attainable with these algorithms, but it is reasonable to think, that for this problem of anomaly detection, considering that the training that must done on a small device, the GMM is the solution to choose.

3.4.2. Kind of anomalies detected

We see that the algorithms are good at detecting point anomalies but have virtually no performance for the other kinds of anomalies, contextual and collective. We have not tried to make any specific adaption of the algorithms to these kinds of anomalies. But it is clear that:

- Contextual anomalies, here where the context is the time, could be detected if we would use specific models for specific times of the day (the day is divided in three of four periods for instance). However, the learning time would be much longer.
- Collective anomalies could be handled here using the integration of the features on different time periods. This would help to detect a null consumption if the normal consumption is generally not null. However, this would entail a large delay in detection.

4. Usage examples

4.1. Problem to solve

Users are interested in home systems that will report events, mostly abnormal ones. For example, an un-expected sound inside their home could be reported (glass breaking …).

4.2. Existing solutions

4.2.1. Pre-configuration requirement

Existing "security" or "elderly care" systems already properly report events but are limited to detecting pre-configured situations that have been programmed by using, for example, a scripting language.

By default, all "non-pre-programmed" situations will be either ignored or reported, by default, as abnormal.

4.2.2. Cloud vs. Local processing

Existing solutions generally use cloud-based processing, local HW is limited to sensors + sensor interface to the cloud (backend). This is for example the case for the "Hive" solution. Note that this solution is extremely intrusive in term of privacy as, for example, 24/365 sound capture could be sent to the cloud for recognition.

We provide LOCAL processing thus protecting privacy and limiting the volume of data flowing to the backend. This solution might appear as more costly because it requires a larger CPU. In fact, modern GW or STB or IoT-dedicated SoCs are now powerful enough and can run neural networks algorithms locally.

4.3. Proposed approach

The system will use AI to learn from a known-safe situation during a long period of time. A model will be created by polling all the sensors that are available. Notice that only the anomaly detection model is trained at home. The acoustic classification model is trained in lab by the manufacturer.

When the model being created is considered to be good enough, all the sensor data will be monitored and any data that drifts from the "normal behavior" will be reported as "abnormal".

This two-phase-model (learning / operational use) will be continuously updated with new data.

The system is able to report events that are outside of the automatically learned "normal situation" and will do that without any user-level configuration or programming (no scripting language).

The system also proposes a more complex sensor dedicated to the recognition of exceptional sounds. This brick uses neural networks technology and requires 10's to 100's of thousand sounds to create a good enough model. It is not possible to learn from user inputs because of this huge data requirement, it is also not possible because, for some sounds, it is not possible to ask the user to produce the targeted sound (glass breaking …).

4.4. Use cases

4.4.1. Sensors

The system is sensor-agnostic, any continuous variation data can fit. In the following use cases, we will consider these sensors:

- Temperature
- Light (color & level)
- Atmospheric pressure
- Sound level (sound pressure)
- Humidity
- Organic volatile gas
- Presence detector (passive infrared detection, PID)
- Geophone (very low frequency microphone (< 100Hz), vibration detector). OPTIONAL (expensive)
- (time / date): not really a sensor but a crucial element.

4.4.2. Audio recognition

We will also use the output of the audio sound recognition brick that can detect pre-trained abnormal sounds:

- Dog barking
- Glass breaking
- Gun shots
- Baby cries
- Alarm (CO, fire …)
- Human voice (no speech recognition, just "someone is speaking" information)

4.4.3. Actors, entities

- NN Audio recognition + pre-trained sounds models recognize rare sounds (dog barking, glass break …). The NN model is prepared offline (require huge computing capabilities)
- Sensor interface: drivers
- Unsupervised learning + NN model: local learning model aimed at detecting abnormal situations
- Post processing: prepares the data, does filtering if needed
- Backend interface: interfaces to the cloud
- (sensors)
- (clock): time & date

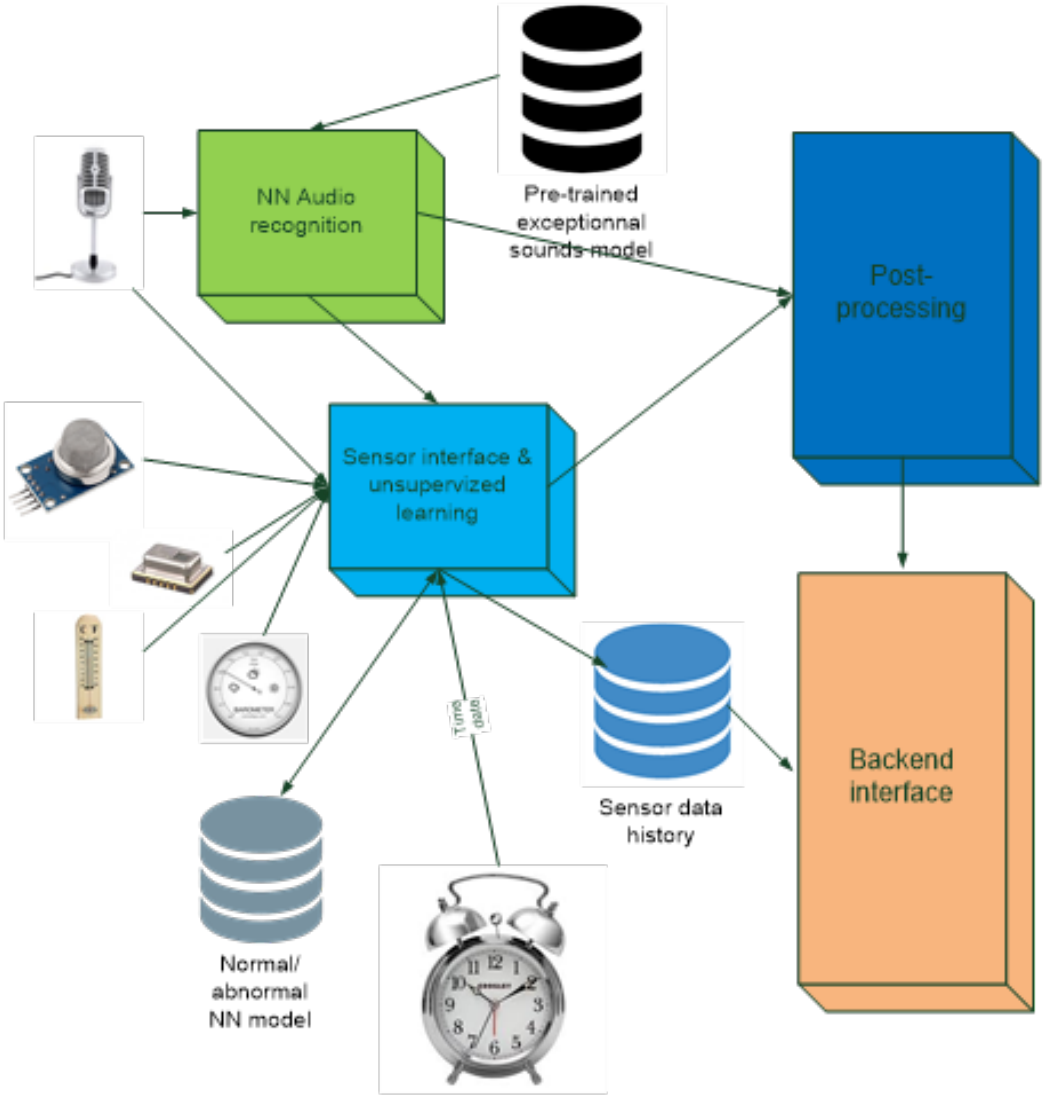

Figure 15 simplified system-level architecture

4.5. Possible usages

Table 4 : Possible usages

What?	Detected as ...	Comments / limitations
Fire detection	Very fast temperature rise High temperature is reached, for instance above 50°C And/or Fire alarm sound detection	Direct sunlight on the sensor might be detected as fire and can happen only a few days per year (high seasonality)

Freeze warning	In case of heating failure, the temperature falls under 5°C. There is a danger of frozen pipes in the house.	
Flood	Very fast humidity rise	
Device reliability	Magnetometer / gyroscope: device is moved	If the device is moved, its previous learning is not relevant anymore
Open window Heater / AirCo issue	Temperature rise / fall above limits – maybe with a humidity modification at the same time	
Dog barking for too long	Dog barking detection + time	Dog barking from the sound recognition brick filtered with "normal" duration by the unsupervised learning
Glass breaking, gun shots, fire alarm sound	Rare sound recognition	No filtering
Baby cry for too long	Rare sound recognition + time	Filtered by the unsupervised learning / time
Presence at wrong time	Presence detector + time + unsupervised learning	Presence detection at unusual time/date
Smoke / CO^2 / Gas	Smoke detector (if any) and/or alarm sound detection	No filtering?
Abnormal noise level (too loud, too long), for example, faucet kept open	Sound energy detector	(not managed by the sound recognition) Detected by measuring sound energy (average), any loud and lasting sound will be recognized the same way
Door slamming	Pressure and/or loud sound	Abnormal pressure rise / fall
Shutters not opening or opened at the wrong time	Light / light color detector	

Light kept switched on at the wrong time	Light detector + time/date	

4.6. More difficult usages, Geophone-related & others

Table 5 : Possible usages (2nd level)

What?	Detected as ...	Comments / limitations
Earthquake / explosion	Geophone activity	The geophone is optional
Steps	Geophone activity	The geophone is optional
Steps at the wrong time	Geophone activity & time	Steps at the wrong time (night?). Geophone is optional
Everything related to power (mains) usage	Currently, no sensor	Could be easy to use, needs adding a dedicated sensor (or set of sensors) on mains
Abnormal steps	Geophone activity	The geophone is optional. "abnormal" steps might be "too loud", "too slow", "too fast" …

5. Possible Implementations

The recognition bricks have been designed in order to run on relatively low-performance hardware while keeping the important goal of running all the AI software locally.

Simplified requirements are:

- CPU: between 50 and 100% of one 32 bits ARM core (Cortex A53 class, ARMV7 instruction set). Note that most A53 implementations have 2 or 4 cores.
- Linux or Android. 256~512MB extra RAM required
- Between 2 and 4GB of storage / code space

Using 64 bits implementation (ARMV8) can improve the porting, having access to a TPU (or a GPU) can also help but it is not a strong requirement.

Several implementations are possible:

- Run the SW bricks inside a CE equipment, for example a set top box
- Run the SW bricks inside a gateway
- Create a dedicated standalone accessory
- Run the SW at the Edge / inside the cloud

Table 6 : Possible implementations

Solution	Pluses	Minuses	Potential improvement / remarks
Set Top Box	Cheap. STB CPUs are fast enough. Ample RAM/Flash.	Not necessarily well located. Might be switched off at random time. Sensors might be missing	Add remote mics
Gateway	Very cheap (but remote mics/sensors will add cost)	CPU & RAM might not be enough. Remote mics are required (can be costly). Always ON device.	Dedicated USB key (including CPU, RAM and sensors/mics)
Standalone device	Very convenient, multiple devices might cooperate. Sensors can be part of the accessory.	Somewhat expensive (but not so much compared to remote sensors + mics devices).	One intelligent accessory connected to simpler sensors boxes.
Cloud / Edge	No CPU/RAM/Storage limit while being cheap	Still need local sensors/mics (expensive). Does not meet the privacy constraint	Less privacy protection is a major issue inside the EU (GDPR). Privacy protection is one of the main differentiators of the technology.

6. Conclusion

The technology we developed allows seamless anomaly detection and reporting for many home-based situations.

Privacy is preserved because no data is sent outside of the home, with only event notifications uploading to the cloud.

Exceptional sounds detection is done using world class AI with models that are created using thousands of sounds for each class, thus achieving extremely good recognition levels.

Installation is simplified because the system can learn by itself from a “normal” situation and then can report any event that is not sufficiently in line with this situation.

Implementations can be either as a standalone accessory or as a SW brick running in existing CE-grade equipment.

Abbreviations

ML	Machine Learning
RMS	Root Mean Square
PCEN	Per Channel Energy Normalization
FFT	Fast Fourier Transform
SVM	Support Vector Machine
NN	Neural Network
CNN	Convolutional Neural Network
FC	Fully Connected
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
RBF	Radial Basis Function
EM	Expectation Maximization

Bibliography & References

[1] Distance measures for speech recognition, psychological and instrumental, Paul Mermelstein, 1976

[2] Trainable frontend for robust and far-field keyword spotting, Wang, Y., Getreuer, P., Hughes, T., Lyon, R. F., & Saurous, R. A. (2017, March). In Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on (pp. 5670-5674). IEEE

[3] SoundNet: Learning Sound Representations from Unlabeled Video, Yusuf Aytar, Carl Vondrick, Antonio Torralba, 30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

[4] DISCRIMINATE NATURAL VERSUS LOUDSPEAKER EMITTED SPEECH, Thanh-Ha Le1;2, Philippe Gilberton1 and Ngoc Q. K. Duong, Technicolor and Eurecom, ICASSP2019, Brighton, UK

[5] Anomaly Detection: A Survey, Varun Chandola, Arindam Banerjee and Vipin Kumar, ACM Computing Surveys, vol. 41, no. 3, July 2009.

[6] Individual Household electric power consumption data set, Georges Hebrail, and Alice Berard, in UCI Machine Learning Repository,
https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption

[7] A.P. Dempster, N.M. Laird et Donald Rubin, « Maximum Likelihood from Incomplete Data via the EM Algorithm », Journal of the Royal Statistical Society. Series B (Methodological), vol. 39, no 1, 1977, p. 1–38 (JSTOR 2984875)

