Doctoral Thesis

Lille University of Science and Technology

Automatic Detection of Screams and Shouts in the Metro

Author:

Pierre LAFFITTE

In fulfillment of the requirements for the degree:
Docteur en Traitement du Signal et de l'Image

13 December 2017

Thesis committee:

Geoffroy PEETERS - Senior Researcher at IRCAM - Reporter

Slim ESSID - Professor at TelecomParisTech - Reporter

Laurent Daudet - Professor at Paris Diderot University - Examiner

Corinne Fredouille - Associate Professor at Université d'Avignon et des Pays du Vaucluse - Examiner

Laurent Girin - Professor at Grenoble-INP - Thesis Director

Charles Tatkeu - Professor at IFSTTAR - Thesis Director

David Sodoyer - Associate Researcher at IFSTTAR - Thesis Advisor

Contents

Chapter 1

Introduction

The works presented in this thesis explore a specific area within the field of intelligent surveillance systems for public places. Such systems have been of increasing interest in the past decades, owing in part to growing safety concerns, and to the developments of artifical intelligence and pattern recognition. From video surveillance of public places to signal intrusion and circuit anomaly detection, artificial intelligence has been used if not as a replacement at least as an aid for humans, to monitor the environment and detect abnormal situations. Generally, automatic surveillance systems use the signal acquired from one or more sensors to gather information about the environment in order to automatically make decisions. Video sensors are the most common tools to date, but are not without their faults. The obstruction of the field of vision, potential changes in lighting conditions, are two examples of the limitations of such systems. As a complement, audio offers robustness to those conditions as it inherently does not suffer from the same issues. Some studies have already explored the combination of audio data with data from other sensors, such as video data, and some have started exploring the benefits which could be drawn from the use of audio signals in the framework of surveillance, in different fields of applications such as home intrusion or security in public places. The implications of the type of environment considered are important; the energy level and the variability of the surrounding acoustic environment can be starkly different.

This study is set in the context of public transportation, and focuses on an embedded approach, which entails a very particular type of acoustic background which we will present in details later. The objective is to build a system which can automatically detect violent situations in a public transportation system (metro, tramway, etc..), relying on the analysis of the acoustic environment to detect screams and shouts when they occur. To this end we will investigate the use of AI techniques for the analysis of audio signals, within the scope of Neural Networks which will be trained to detect violent situations through the analysis of the audio environment. AI has been used in many ways to analyze audio signals for decades, as the promincence of smartphones can attest; virually all smartphones nowadays are equipped with a voice recognition tool based on AI. Additionally the music industry also shows signs of a growing interest in AI, as its ability to automatically extract and gather information about the music opens up a vast scope of perspective to music providers, the most common applications being music recommender systems, automatic playlist generation, and music identification and recognition [17].

To set the stage, we will outline the basic principles of pattern recognition and how it can be used in the context of audio event detection in Chapter 2. We will break down the problem and see how it can be defined from a classification perspective. The characteristics of audio signals and their representation in the context of classification will be presented and analyzed, as well as the existing methods which can be considered for the task, and their theoretical background.

In Chapter 3 we will present the mathematical framework underlying Neural Networks, which will be the basis of the classifiers used to perform our task. We will expound the basis of Linear Models and derive their probabilistic implications, leading to the presentation of the Logistic Regression unit which is the building block of Neural Networks. We then show how the latter use this building block to achieve more complex architectures. Details of different such architectures will be provided (NN, CNN, RNN), outlining their use in audio pattern recognition.

Chapter 4 provides a detailed description of the database, collected in a real environment, with realistic simulations of our target application. The realistic conditions will be detailed and their impact on the audio environment will be analyzed. We will show how the resulting database is challenging in its diversity and noisiness, with numerous concurrent sound sources and high energy. A detailed description of the various elements composing it will be provided, in which we

describe the different forms of occurrences of our main target events, screams and shouts, as well as other vocal events, and their relation to the acoustic background environment.

In Chapter 5 we consider the problem as a 3-class task, where the classifier has to predict a voice-related class out of the following set: {'shout', 'speech', 'background' }. For this task we will conduct preliminary tests to identify the best feature representation, and the best network configuration for each of the 3 ANN architectures presented in Chapter 3 (NN, CNN, RNN). For each experiment we provide a detailed description of the settings used, before giving the results and drawing conclusions on them.

Based on these conclusions we turn to a different approach in Chapter 6, with a view to overcome the difficulties identified in Chapter 5 by taking into account the acoustic background environment explicitely through the addition of different classes. We consider a second labelling scheme which describes the acoustic background environment, and use the classes from that environment to create compound classes with the set of classes used in the previous chapter. We end up with 15 classes, carrying each a description of both voice-related content and environment-related information, on which we perform a 15-class classification.

In a last attempt to account for the agressive acoustic background environment we adopt a multi-label paradigm in Chapter 7, where all input data are described with two different labels which the network needs to predict. In this paradigm the classifier learns in a joint fashion both voice-related classes and environment-related information, using its modelling power to predict both at the same time.

Finally Chapter 8 will summarize the main aspects viewed in this thesis. We will recall the results obtained with the proposed methods, and draw conclusions as to the relevance of those methods to the task, as well as the lessons we can derive from it. In line with these conclusion we will suggest some ideas for future works and lay out some perspectives.

Chapter 2

Automatic Sound Classification: General principles and state of the art

As stated in the introduction, this work deals with automatic detection of screams and shouts in the subway environment. In this chapter, we present the foundations to address this kind of problem. We start from the general principles of the pattern recognition problem and methodology to address it, and along the way, we progressively introduce the specific case of sound detection and recognition. In the second part of the chapter we complement the start of the art by presenting a series of existing works and applications in related domains.

2.1 Fundamentals of Pattern Recognition

The automatic detection of specific sounds implies the recognition of audio patterns. This naturally leads to Pattern Recognition, a branch of Machine Learning rooted in Engineering, whose objective is to provide a mathematical solution to the recognition of recurring patterns embedded in raw data. In the following, we outline the general paradigm of Pattern Recognition and its main areas, before delving into its application to audio signal.

The broad objective of Pattern Recognition boils down to the assignment of a prediction to an observed input. This prediction can be a pre-defined label which can represent classes, in which case the task is called Classification, or any continuous real value, in which case the task is called Regression. In our case, since we know precisely what we want our system to detect, we can describe it with a pre-defined label. Therefore we will focus solely on Classification here.

In practice, making a prediction about some given data requires to take into account the context around the data, which is why we may distinguish the inference stage where the model outputs a mathematical result based on its theoretical setting and on the data, and a "final decision" stage where this result is tempered by a cost before the final decision is made. This cost parameter represents the burden of making a particular decision if it were wrong, in terms of applicative results. A self explanatory example often given is that of a cancer diagnosis: falsly diagnosing a patient with cancer leads to no more than an undully fear, whereas not detecting a cancer in a sick patient has much more serious consequences. The cost of the second mistake should clearly outweigh that of the first one, because it is a much less acceptable mistake. The present work focuses solely on the inference stage and does not intend to investigate the "final decision" stage.

2.1.1 Learning-Based Classification

Figuratively, the aim of Classification is to split the data into separate groups and put them in different boxes, where each group is associated with a pattern [30]. All models introduced in the following are data-driven, meaning they learn progressively how to classify data by analyzing data. This process is generally divided in two steps: a training stage and a running stage. A dedicated dataset is used for the training stage, allowing the model to learn the general structure of each group. The knowledge acquired during the training stage is then used in the runing stage, when the classifier is provided with new unseen data, to find the most suitable box. In more concrete terms, the boxes are called classes. When the class labels of the training data are known, the learning process is called supervised. Otherwise, it is called unsupervised.

In a classification paradigm (as opposed to regression), after being trained on the training dataset, the system provides an estimate of (or a parametric model of) the posterior probability of each class $c \in [1, C]$, where C is the number of classes, given a new input data vector $\mathbf{x}_n$:

$$\mathbf{y}_n = \{p(c \mid \mathbf{x}_n)\}_{c \in [1,C]}, \tag{2.1}$$

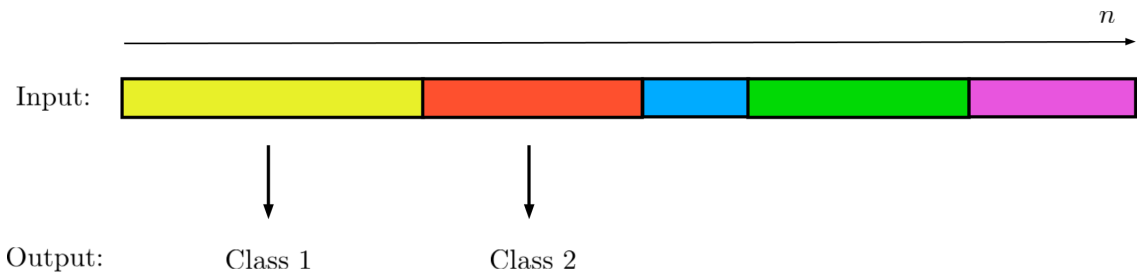

Figure 2.1: Classification of input patterns.

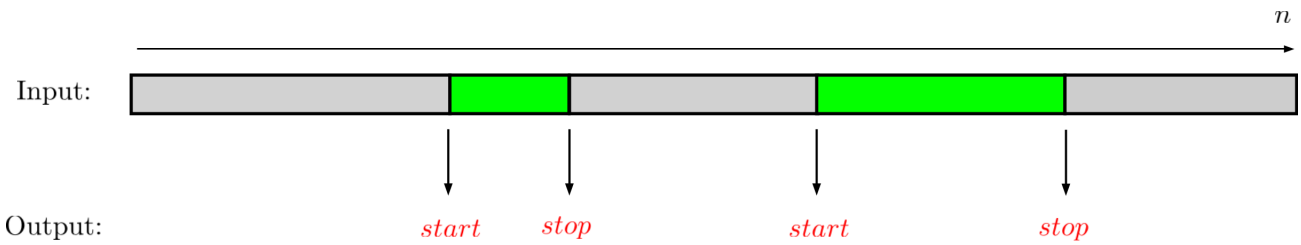

Figure 2.2: Detection of patterns inside an input stream.

where $\mathbf{y}_n$ is the output vector consisting of the posterior probability of each class c, and n denotes an index for the input/output sequence, which is often a time index (that will be the case for sound classification). The selected class is the most probable (the one with the highest posterior probability), i.e the class corresponding to the highest value in $\mathbf{y}_n$:

$$\begin{aligned} \hat{c} &= \arg\max_c \left(p(c \mid \mathbf{x}_n)\right) \\ &= \arg\max_c \left(\mathbf{y}_n\right). \end{aligned} \tag{2.2}$$

During the training stage, classification models optimize their parameter values by minimizing a cost function which represents some average classification error over the complete training dataset. This is usually achieved by means of a gradient descent algorithm which sequentially updates the parameters until it reaches a minimum value of the cost function.

Classification and Detection. A distinction should be made between Classification and Detection; both rely on the use of a classifier but the difference between the two lies mainly in the way the output of the classifier is dealt with. The goal of Classification is to associate an output class to an input, whereas Detection aims to identify when a certain event occurs within an incoming stream, as exemplified in Figures 2.1 and 2.2. The identification of the moments when the event starts and stops is the main difference between the two. Yet the two processes are closely related. In some cases, Detection is used to perform data segmentation before Classification is applied. In some other cases, Classification is directly used in the process of Detection [112], see Section 2.1.2.

Input definition. In order to achieve these objectives, it is necessary to define the input $\mathbf{x}$ of these classification/detection systems, especially for temporal signals. Commonly, features are

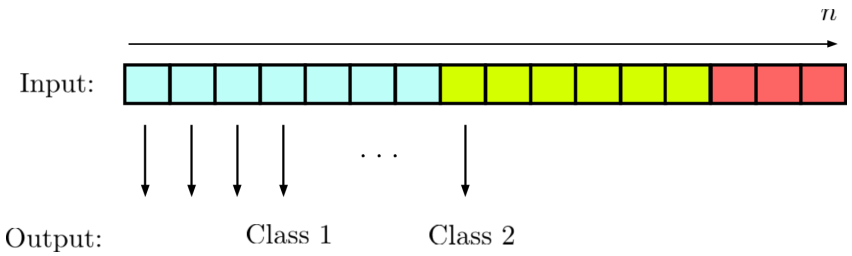

Figure 2.3: Frame-wise classification of input into output.

extracted from a portion of the signal to be analyzed [92]. Such portions (usually referred to as 'frames' in audio signal processing) generally have a maximum duration equivalent to the duration of the stationarity of the signal. Thus, the signal is divided in successive frames with same duration, possibly overlapping, where feature vectors are extracted, thereby defining the input $\mathbf{x}$ of the classification system.

2.1.2 Classification of sequences

Frame-wise classification. Frame-wise Classification refers to the case where each input vector is processed individually, i.e. considering each input vector independently of the rest of the input vectors. The classifier produces a corresponding output regardless of past data (and future data in offline / non-causal processing) and past results, thereby dismissing any connection between consecutive inputs. This principle is illustrated in Figure 2.3.

Sequence processing. Sequences are a particular type of data in that they consist of ordered frames, which can be uni-dimensional (sequence of scalar values) or multi-dimensional (sequence of vectors; we limit the discussion to the case of fixed-size vectors). They stem from processes evolving in time, with very specific time signatures expressed through the order in which the successive features are arranged. In order to fully 'observe' a sound it is generally necessary to capture it throughout its entire duration because the essence of a sound often lies in its evolution through time. Indeed, many key characteristics that allow us to distinguish between distinct sounds are time-related (e.g. the attack-decay-sustain-release shape of a note produced by a musical instrument, pitch, envelope, etc..). The temporal dimension therefore often plays a significant role in sound identification (by Humans and machines) and needs to be taken into account in the analysis process. However in this context, frame-wise classification fails to do so. An idea in trying to circumvent this issue is to add temporal context directly into each input vector, in order to force the model to take temporal information into account without drastically changing the structure of the system. This process is called Context Dependency.

Context dependency. The idea is to consider several consecutive frames over a certain period of time, and concatenate the corresponding per-frame vectors to build one unique input vector. This way, the system is fed with information spanning a certain amount of time. We define the following vocabulary and notations:

- Context Window: time-span covered by the consecutive frames to form the input,
- Context-Dependent Vector (CD vector): 'new' input vector resulting from the concatenation of several consecutive single-frame input vectors,
- T_w: length of the frame analysis window in seconds,
- T_h: frame analysis window shift in seconds; If the frame shift is lower than the frame length, there is some overlap between two (or more) consecutive frames, meaning that some part of the information is shared between succesive frame-wise inputs.
- T_{cd}: length of the context window in seconds,
- K: length of the context window in number of frames,
- D: size (dimension) of each individual single-frame input vector,
- S: size of the CD vectors.

T_{cd}, K and T_h are linked through the following relation:

$$K = \frac{T_{cd} - T_w}{T_h}. \tag{2.3}$$

At each time step n, K previous and current (and possibly future) frames are gathered within a context window and concatenated together, as shown in Figure 2.4. The size of this resulting vector is thus K times the size of the single input vector:

$$S = KD. \tag{2.4}$$

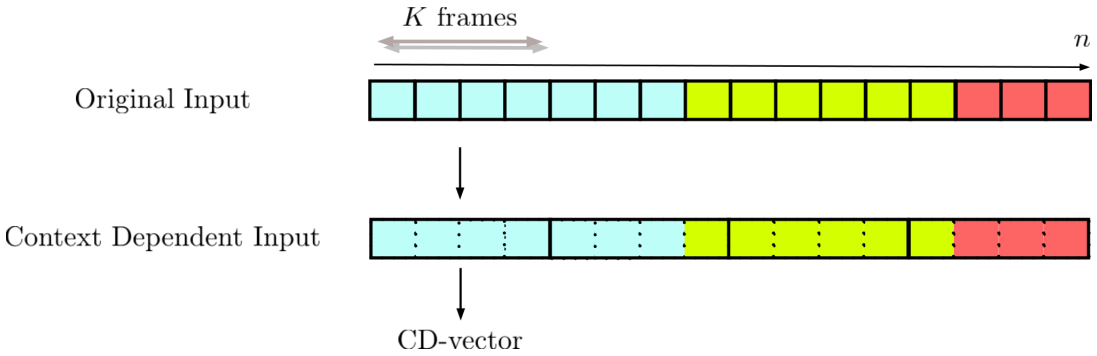

Figure 2.4: Single-frame vectors (top) and Context-Dependent vectors (bottom).

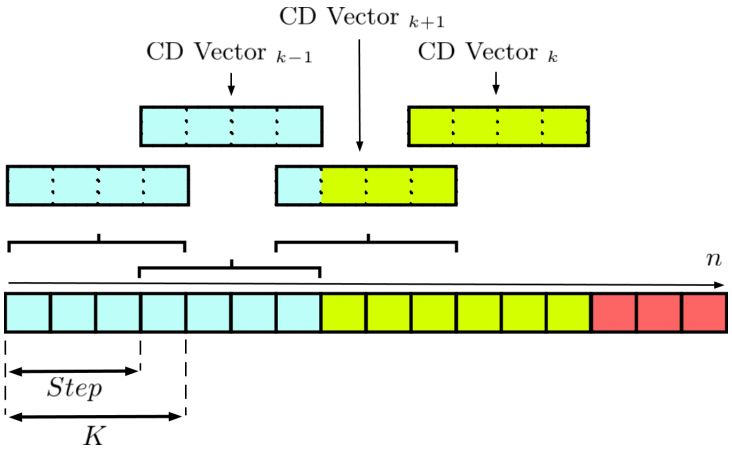

Figure 2.5: Overlap between successive CD vectors.

Step size and redundancy. There are different ways to build the context-dependent vectors across the dataset, but a step size is generally used to define how the context window should slide over the input frames. A step size lower than the context window size allows to smooth the variations across successive CD vectors, introducing some redundancy at the CD window level, as shown in Figure 2.5 (note that this redundancy adds to the possible redundancy at the frame level, if there is some overlap at the frame level). For example, if the step size is 100 ms and the context window size is 1 s, each super vector spans 1s but is separated by only 100 ms from its predecessor and its successor (measured from a common position in the two vectors, such as the beginning, the middle or the end). This amounts to having 10 super vectors beginning within one second, with an overlap of 90% between them. This means that the super vectors have 90% of similar content, leading to a high amount of redundancy. With a 500 ms step size and a 1 s context window, the overlap goes down to 50%. The redundancy can be avoided by setting a step size equal to the CD window size, but in general, a reasonable amount of redundancy is desirable to smooth the system output sequence and associated decision process (for classification problems). Moreover, overlapping between successive CD windows enables to increase the size of a given dataset without additional data.

Note that if the context window shifts by one frame, the classifier makes one decision for each new input frame, just as in frame-wise classification. The difference with frame-wise classification then lies is the fact that the decision is made with more context awareness. If, however, the context window shifts by more than one frame, there are less classification decisions than the original number of frames (before windowing). This is not a problem since the classifier outputs the class that it believes matches the sub-sequence of frames within the context window. In that case, the step size really controls the amount of redundancy across successive context windows, and the data may now be considered as a sequence of context windows, rather than the sequence of original frames.

Context dependency and sequence classification for audio signals. Within the framework of audio event detection, the target patterns typically are sequences describing the temporal evolution of the sound spectral content. Frame-wise input vectors are thus vectors of spectral coefficients

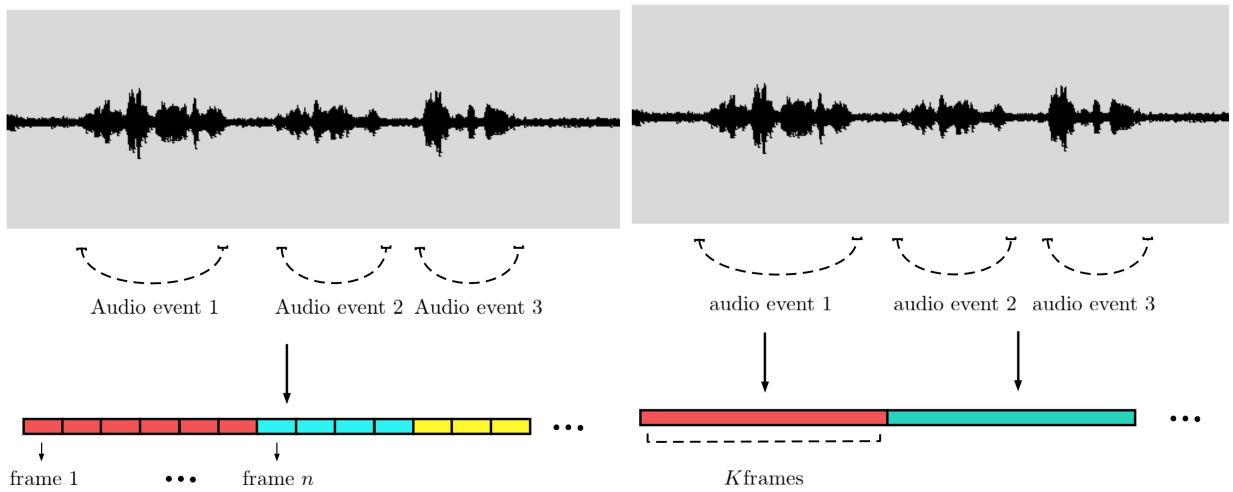

Figure 2.6: Left: Three successive sounds are represented by consecutive frames considered as independent one from another. Each color stands for a different sound event. Right: The same sounds are represented with Context-Dependent vectors of K frames. In this example, the size K of the context window is too large with regards to sounds 2 and 3 which are wrongly considered by the classifier as a single event.

representing the short-term distribution of the signal energy as a function of frequency. Typically, the coefficients are modulus square of Discrete Fourier Transform, i.e. an estimate of local power spectral density (PSD), or mel-frequency cepstral coefficients (MFCC), representing the log-scale spectral envelope with a mel-scale frequency axis [111]. Each vector of spectral coefficients is estimated using a time window of size T_w in the range of 10 ms to, say, 60 ms (typically 30 ms) sliding along the signal, generally with some overlap (typically 10 ms) [3, 16, 31]. As stated before, sounds are generally (much) longer than a few tens of ms. Consider the sound of thunder for example; it usually breaks over a few seconds of time, and during the lapse of time before the deep ending sounds occurs, it could be mistaken for a lot of things (cracking beams inside the house, rumbling of a mechanical engine, etc..). This is a perfect example of a succession of different short-term audio patterns (spectro-temporal variation) defining one audio event. Listening to only 30 ms of thunder can be very deceiving if the goal is to identify the source of the sound. To remedy this, Context Dependency, as described earlier, allows the classifier to consider the temporal evolution of sounds over a given duration (defined by the length of the context window). However, the main drawback of this approach is that it defines a fixed context window size of K frames, when different sounds can have (and generally have) different length, and often very different length. The greater the value of K the more chances that two consecutive sounds be captured within the same CD vector. Figure 2.6 illustrates the influence of the number of K frames in the process of creating CD vectors. As a compromise, it is generally better to choose K so that a CD vector is at most as large (on the temporal axis) as the smallest sound event to be recognized, in order to minimize the mixing of successive events in a CD vector.

Fixed-sized input machine learning models are relatively easy to implement but they have to face that problem when applied to variable-length temporal signals. Some groundbreaking methods were introduced in order to handle signals of arbitrary length such as Dynamic Time

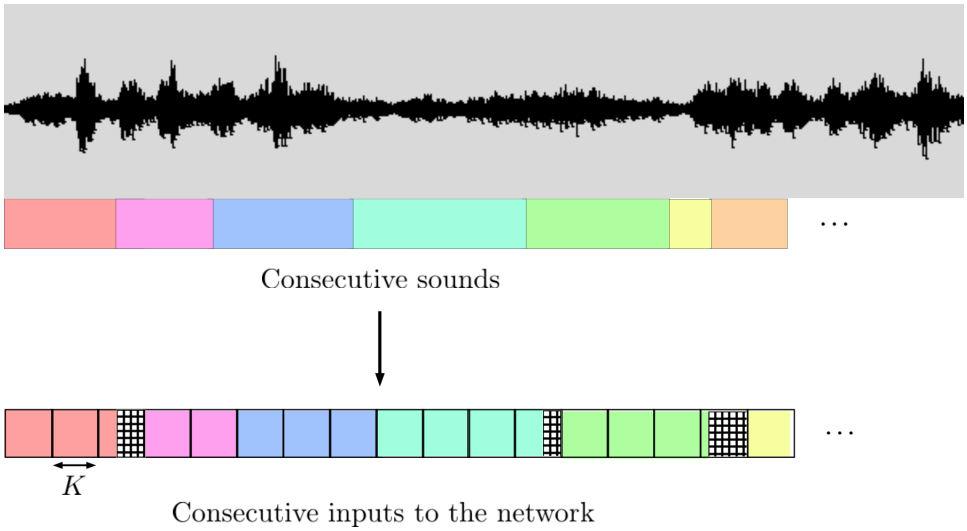

Figure 2.7: Data formatting following the pre-segmented mode. Different colors indicate different sounds. Hatched regions indicate zero-padding. Note that in this figure (and the following) an elementary compartment of input data represents a CD frame of K input frame-wise vectors, and not one frame-wise vectors as in the previous figures.

Warping [105] and Hidden Markov Models [99]. While DTW can not actually deal with arbitrary length signals, it does allow for variation in length, up to a certain extent. HMMs on the other hand can theoretically deal with infinitely long sequences, but they have their drawbacks. HMMs will be described in Section 2.1.4.

Streaming vs pre-segmented. In the literature, automatic sound classification is often performed and evaluated offline with pre-segmented audio patterns (possibly completed with zeros, i.e. a zero-padding pre-process at sound boundaries). In the case of context dependency configuration, this entails that the K frames composing the CD vectors are all from the same sound. However in the present framework of alert signals detection, we have to perform an on-line analysis on an incoming "continuous" audio stream, where the sound patterns that we must detect are thus no longer pre-segmented. In this case, a frame of signal can contain parts of two different sounds and concatenating the K "current" consecutive input frames (for instance the current input frame and the $K-1$ past frames) does not necessarily match the onset and offset of the sound patterns: In the neighborhood of class boundaries (when a sound segues into another sound from a different class), a CD vector can contain vectors from the first sound concatenated with vectors from the second sound, which naturally brings some ambiguity in the classification process (test time) and/or the estimation of the models (training time). In that case, in the present work, the label of a CD vector containing segments pertaining to two different classes is chosen to be the class with the highest number of frames within that supervector. This way of formatting and processing a continuous stream data will be referred to as *streaming* (as opposed to *pre-segmented*). Figures 2.7 and 2.8 illustrate the difference between the two data formatting processes. In Figure 2.7 the hatched regions represent areas of the input vectors that are padded with zeros in order to match the fixed input size when the sound events do not fit the context window size K exactly. In Figure 2.8 when

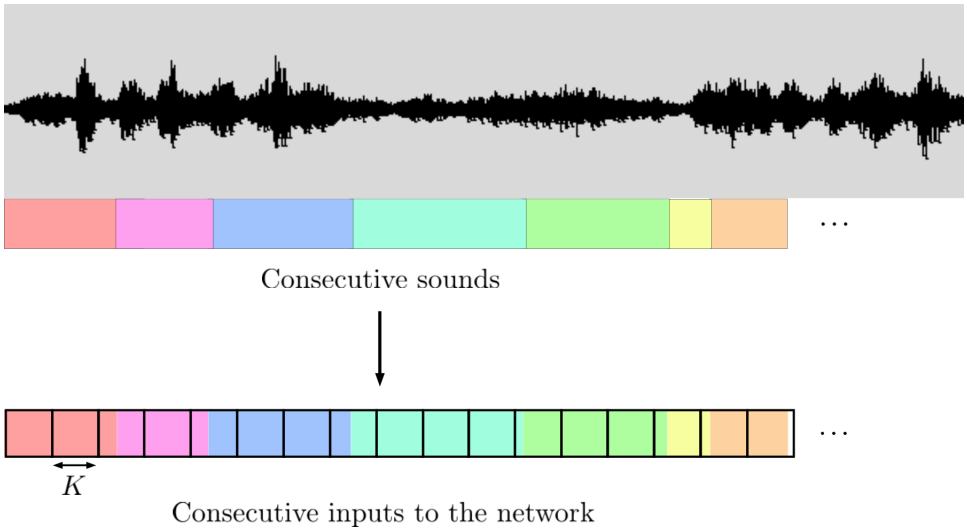

Figure 2.8: Data formatting following the streaming mode. Different sounds can be mixed up within a same input.

the same situation occurs (where a sound does not fit K) a piece of adjacent sound is used to pad the current input block, instead of zeros. As a result we can see that consecutive sounds can end up mixed together inside an input block, which is naturally expected to make the classification more difficult, as will be verified in our experiments on scream/shout detection in the forecoming chapters.

2.1.3 Multi-label Classification

So far we have considered the case where each input belongs to one single class, because we relied on the assumption that the classes were mutually exclusive, i.e sounds from different classes could not occur at the same time. However in many real-case scenarios, one may be confronted with the situation where that assumption is not true any more, and sounds from multiple classes can occur at the same time. In that case, the input data needs to be associated with multiple target classes, which is not without consequences on the classification paradigm. This is referred to as multi-label classification as opposed to single-label classification [90, 118]. The classification equation (2.2) for single-label now becomes, for multi-label:

$$\hat{\mathbf{c}} = (\mathbf{y}_n > \gamma) \tag{2.5}$$

where $\hat{\mathbf{c}}$ is the vector containing the estimated classes (potentially multiple in the present multi-label framework) and the right-hand side expression $(y > \gamma)$ is a boolean expression returning a boolean 'True' value at all indices where the predicted probability is greater than the threshold γ. In this new paradigm, the classifier has to be able to learn from multi-class training examples, and to output multi-label predictions, as shown in Figure 2.9. The former aspect is addressed by considering the target classes as separate random binary variables (which can thus be on at the same time, indicating that the corresponding events are detected/occurring simultaneously) and then deriving a suitable cost function according to those variables.

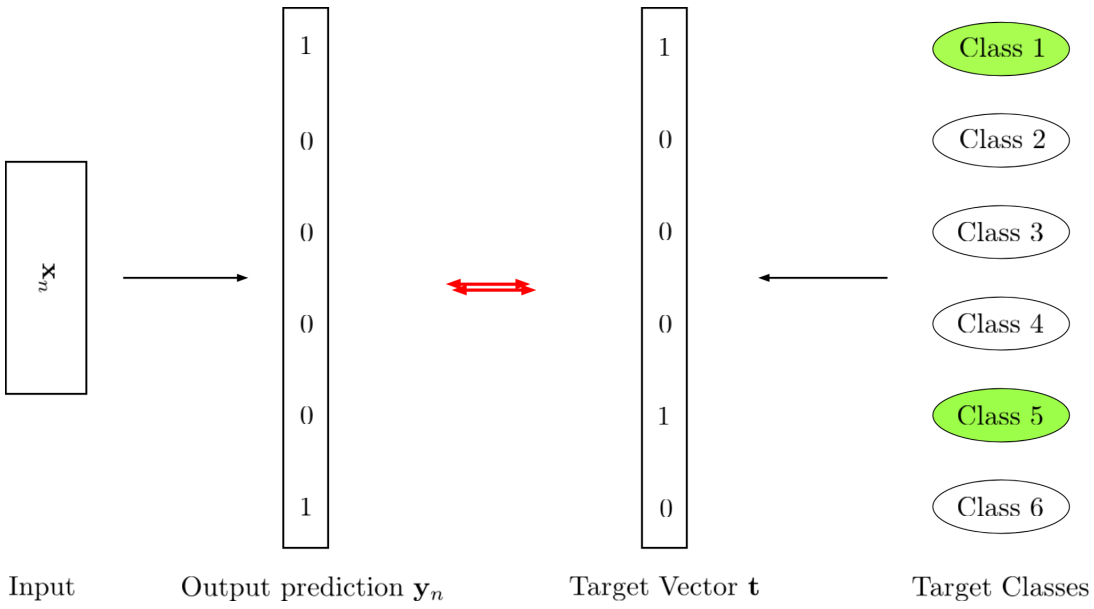

Figure 2.9: Illustration of the multi-label classification principle in a 6-class case.

2.1.4 Generative Models

We have seen how the audio input signal can be represented in the classifier and pre-processed, and we have raised some of the issues related to it. We now turn to the models which will be fed with these inputs, in order to generate a prediction according to Equation (2.2). As defined in this equation, an input feature vector $\mathbf{x}$ and a statistical model $p(c \mid \mathbf{x})$ for each class c are required. For the estimation of $p(c \mid \mathbf{x})$ two main types of models are defined: Generative models and discriminative models. We rapidly present the generative models in this subsection and we will deal more extensively with the discriminative models in the following of the chapter, since we used the latter in our work.

In a classification setting, generative models attempt to model the distribution of the observations $\mathbf{x}$ along with the distribution of the target class c, in order to differentiate between classes [10]. Concretely, they use Bayes' rule to compute the class posterior $p(c \mid \mathbf{x})$:

$$p(c \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid c)p(c)}{\sum_{b=1}^{C} p(\mathbf{x} \mid b)p(b)} \tag{2.6}$$

$$= \frac{p(\mathbf{x}, c)}{\sum_{b=1}^{C} p(\mathbf{x}, b)}, \tag{2.7}$$

where we remind that C is the total number of classes. This amounts to modelling the joint distribution of the data and classes $p(\mathbf{x}, c)$, which implies that new synthetic data could be created by sampling from this distribution, hence the name Generative. In principle, $p(\mathbf{x}, c)$ can be modeled by any probability density model, e.g. the Gaussian distribution. However, it is often the case that not all variables characterizing the set of classes are directly observed through $\mathbf{x}$. These non-observed features call for the concept of "hidden state". By this paradigm, the complexity of each class is embedded in the expression of one state or multiple states, embodying the different

situations (or aspects) which constitute the definition of a class. This entails that only one state can be expressed at a time, meaning that each single datapoint from a class stems from only one state. Generative models considering hidden states include Gaussian Mixture Models (GMMs) [41], Hidden Markov Models (HMMs) [99], Deep Belief Networks (DBNs) [51], etc. In the following we explain the GMM and HMM models, whereas the DBN will be introduced in Section 3.1.3.

Gaussian Mixture Models. Mathematically the GMM belongs to the category of Mixture of Experts, which models a target distribution by a weighted sum of parametric components. Here the components are Gaussian and the expression of a GMM is thus:

$$p(\mathbf{x}) = \sum_{i=1}^{I} p(h=i)p(\mathbf{x} \mid h=i), \tag{2.8}$$

with

$$p(\mathbf{x} \mid h=i) = \mathcal{N}(\mathbf{x}; \mu_i, \mathbf{\Sigma}_i), \tag{2.9}$$

where I is the total the number of states of the variable h and μ_i, $\mathbf{\Sigma}_i$ are respectively the mean vector and the covariance matrix of the Gaussian kernel $\mathcal{N}(\mathbf{x}; \mu_i, \mathbf{\Sigma}_i)$ characterizing state i. $\pi_i = p(h=i)$ are the weights of the Gaussian kernels in the mixture, with $\sum_i^I \pi_i = 1$. All these parameters can be estimated iteratively during the training process by maximizing the likelihood of the data under the model. A typical algorithm used for such likelihood maximization of GMM parameters is the Expectation-Maximization algorithm (EM) [24].

In a GMM, assuming that one state corresponds to one class, Equation (2.7) amounts to:

$$p(c \mid \mathbf{x}) = \frac{\pi_c \mathcal{N}(\mathbf{x}; \mu_c, \mathbf{\Sigma}_c)}{\sum_{i=1}^{I} \pi_i \mathcal{N}(\mathbf{x}; \mu_i . \mathbf{\Sigma}_i)}. \tag{2.10}$$

A GMM thus directly provides an estimate of $p(c \mid \mathbf{x})$ for each class c. This can be easily extended to using several states to model one class. One of the main limitations of GMMs is their inability to deal with sequences, an issue we already mentioned in Section 2.1.2. Nonetheless their association with a temporal model called Hidden Markov Model (HMM) overcomes this issue.

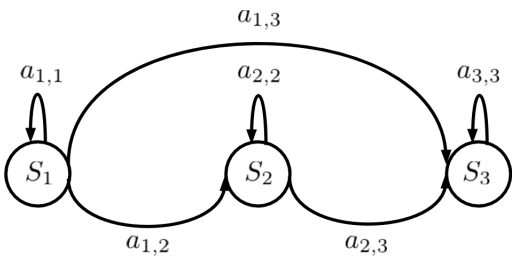

Figure 2.10: Hidden Markov Model with three states.

Hidden Markov Model. An HMM adapts the concept of hidden states to the time dimension by building a sequential model of transitions between states. It thus uses the same concept as GMMs whereby each class is represented by one (or several) hidden state(s), and one observation vector

is associated to each state, with the additional property that the states are now part of a Markov chain, i.e. the state h_n of the model at (time) step n only depends on the state h_{n-1} at step $n-1$:

$$p(h_n \mid h_{n-1}, ..., h_0) = p(h_n \mid h_{n-1}). \tag{2.11}$$

The state transition model is embedded through the transition probabilities $a_{i,j} = p(h_n = i \mid h_{n-1} = j)$ ruling the possible combinations of states in time, as illustrated in Figure 2.10. If we consider the states sequence:

$$\mathbf{h} = h_0, h_1, ..., h_n,$$

and the associated observation sequence:

$$\mathbf{X} = \mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_n,$$

the joint law of $(\mathbf{h}, \mathbf{X})$ is :

$$p(\mathbf{h}, \mathbf{X}) = \prod_n^N a_{h_n=i_n, h_{n-1}=i_{n-1}} p(\mathbf{x}_n \mid h_n = i_n), \tag{2.12}$$

and the distribution of $\mathbf{X}$ is the sum of $p(\mathbf{h}, \mathbf{X})$ on all the possibilities of $\mathbf{h}$, weighted by $p(\mathbf{h})$. The estimation of $p(h_0)$, $a_{i,j}$ and of the parameters of conditional observation distributions $p(\mathbf{x} \mid h)$ is performed by the Baum-Welch algorithm [99]. Note that the transitions between states in the presented HMM model are unidirectional, i.e. the process can not go backwards. This type of HMM is called a left-right HMM and is widely used to represent speech phonemes in ASR [99].

Once the model is learned, the problem of estimating a new state sequence given a new observation sequence is solved by the Viterbi Algorithm. Thanks to a smart recursive computation, this algorithm provides the optimal state sequence over the complete (new) observation sequence, i.e. the state sequence with the highest probability, without having to compute the probabilities of all possible state sequences,. For more details on the derivation of the Viterbi algorithm the reader is referred to [99].

Generative models for audio Classification. Applications of Generative models for audio classification include GMMs used to detect gunshots [19, 38], screams [38] and environmental sounds [16]. As mentioned above HMMs [99] and DBNs [97] have been widely used in Automatic Speech Recognition (ASR).

2.1.5 Discriminative Models

Discriminative models, on the other hand, attempt to calculate $p(c \mid \mathbf{x})$ directly. They comprise Logistic Regression, the k Nearest Neighbors algorithm (k-NN), Neural Networks (NN), Support Vector Machines (SVM), etc. Their aim is to find the best separation (according to some metrics) between all classes, resulting in a division of the feature space into one subspace per class. The difference with the generative approach is that here the information about the precise shape of individual distributions of classes is considered poorly relevant. Instead the focus is on the boundaries between classes; the classifier decides which class the data belong to by looking at their location with respect to those boundaries. In short, Discriminative models attempt to separate the classes by finding the shapes of the boundaries between classes, while Generative models attempt to mimic the statistical process that generated the data. The former are commonly considered to be more efficient on classification tasks [86].

Support Vector Machines. Popular Discriminative techniques widely used are the Support Vector Machines (SVMs) and Artificial Neural Networks (ANNs). ANNs will be at the heart of the present thesis work, hence they be will specifically introduced in the next subsection, and their technical description will be made in more details in Chapter 3. Here, we give a few words about SVMs. As a large margin classifier, the SVM assumes that the classes are linearly separable in some feature space, which is obtained by a non-linear transformation Φ of the observation data, resulting in a global a non-linear classifier. In the feature space, a SVM seeks to find the separating hyperplane with maximum distance to the closest points from each class. Such points are defined as support vectors. SVMs make predictions based on kernel functions, defined with respect to the support vectors $\mathbf{x}_{sv}$:

$$k(\mathbf{x}, \mathbf{x}_{sv}) = \Phi(\mathbf{x})^T \Phi(\mathbf{x}_{sv}). \tag{2.13}$$

Each new data point is evaluated against the support vectors through the kernel function, and a prediction is made with the following expression:

$$y(\mathbf{x}) = \sum_{i=1}^{I} a_i t_i k(\mathbf{x}, \mathbf{x}_i) + b, \tag{2.14}$$

where a_i is the Lagrange multiplier corresponding to support vector i, t_i is its corresponding label, and b is a bias term.

Similarly to GMMs, SVMs are not tailored for sequences, and need to be associated with a temporal model, such as HMMs (as introduced above) [99] or Dynamic Time Warping (DTW) [69]. The latter is akin to the Viterbi decoder in that it computes the most probable path, but considers the instantaneous state with the highest probability at each time step, instead of the joint probability of the entire path. In the realm of audio applications, SVM (discriminative model) were found to be on par with GMM (generative model) in terms of correct classification scores, but they triggered less false alarms [101].

2.1.6 Artificial Neural Networks

Artificial Neural Networks (ANNs) are another popular type of Discriminative models. Because they will be the centerpiece of this thesis, Chapter 3 will give a detailed technical description of

the most popular ANN techniques, and only their general principles will be laid out in the present subsection, in order to give the reader a first broad insight.

Artificial Neural Network is a general term that designate a set of discriminative models for pattern recognition, which have been quite popular for some decades now, and used in several fields of applications. All ANNs have in common that they are based on the combination of elementary processing units called 'neural units' or 'neurons' or 'cells', organized into a network architecture. This network is generally organized in different layers of units. The ouputs of elementary units are connected to the inputs of other units via multiplicative factors called weights. In an ANN, the discriminative information in the input data is thus encoded into the weights of the different layers. Training an ANN on a set of training data consists of setting all the weights values so that the discriminative power of the ANN is maximized on this training set, and hopefully on any test dataset whose distribution is close to the training set distribution, according to the general principle presented in Section 2.1.1.

The main advantage of ANNs is that they can learn automatically from the data without requiring any assumption about their distribution, as opposed to parametric classifiers which are generally based on a Bayesian framework, assuming some prior knowledge about the statistical distribution of the data. Most ANNs can be used for both classification (with discrete class-wise outputs) and for regression or generation (with continuous outputs). In the latter case, another strong point is their quality as universal approximators, as explained in [22], giving them the ability to model any non-linear function in compact subsets of $\mathbb{R}$. In the present work, we focus on the use of ANNs as classifiers.

As will be seen below, the different types of ANNs mainly differ by the network architecture, by the nature of the neurons, and possibly by some additional processing modules within the network. ANNs can be broadly classified into direct feed-forward Neural Networks, Convolutional Neural Networks, and Recurrent Networks. We introduce now each of these types of ANNs.

Direct feed-forward Neural Networks. The structure of a direct feed-forward NNs is illustrated in Figure 2.11. In the following, for simplicity and adequacy with a large part of the literature, this type of ANNs will be simply referred to as Neural Networks (NNs). Such a NN is composed of one or several layers of neurons. Each unit performs a non-linear transformation (called the activation function) on a weighted sum of its inputs, which are the outputs of the previous layer when several layers are used. The first layer, called input layer, is simply the input vector $\mathbf{x}$, and the output layer is composed of output neurons, which compute the posterior probability of each class (one output neuron per class). Intermediate layers between the input and output layers are called hidden layers, and when the number of hidden layers is greater than one, a NN becomes "deep". The units in hidden layers, noted h, thus operate in chain, building more complicated transformations of the input, with lower layers providing insight of the input data to higher layers, in order to aid the final classification process. The idea is that the resulting non linear functions between inputs and outputs can be arbitrarily complex.

Hidden layers. Units in hidden layers are an expression of hidden states, as defined in Section 2.1.4, with the difference that they are no longer exclusive. Indeed, recall from Section 2.1.4 that only one state can be expressed at a time. In NNs on the other hand, the hidden states represented by neurons in hidden layers are elementary units that can be on at the same time. From that perspective they represent hidden factors in the data. These factors are abstractions of the

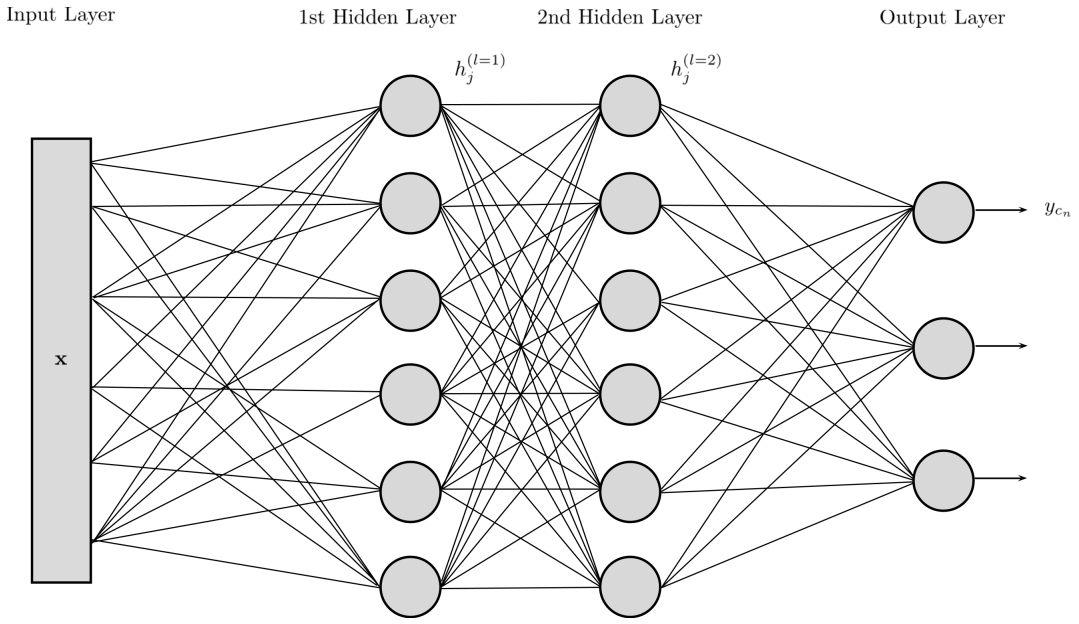

Figure 2.11: A Neural Network with two hidden layers.

data which are similar to what enables the human brain to understand such data. For instance when hearing the sound made by a dog barking, the brain breaks down the acoustic data into several concepts which interact together, enabling the brain to infer some characteristics of the sound (aggressive, repetitive, loud, etc..). This way it can identify that the sound was made by a dog barking, provided it had previously learned to put a label on this type of sound. The characteristics mentioned here are called high-level abstractions: close to human understanding but far from the original data. Low-level abstractions, on the other hand, might not even be fathomable. The more layers, the more conceptual these abstractions become, enabling the network to discriminate the data more efficiently than if it were working with raw data (which is considered the lowest level of abstraction). In a classification task, the target classes are the highest level of abstraction because they are the last level of abstraction that the model achieves.

In light of this explanation on hidden layers, a different perspective can be taken on the operation performed by each individual neuron in the hidden layers (via its activation function) where it can be thought of as an attempt to answer the following question: "Is the input associated with the abstraction or factor associated with the neuron". In other words, the neurons try to unveil certain characteristics of the input by identifying the features which compose that input. Neurons in higher layers are then able to use those features to pursue their own classification goal on it.

History. The first premise of artificial neural networks dates back to 1943 [75], proposing a description of how neurons function in the brain. Several ANN models were then proposed for pattern recognition, using either supervised [100] or unsupervised learning [47]. Deep architectures appeared with [56] and are referred to as Deep Neural Networks (DNN). However it was not until an efficient training algorithm called back-propagation (BP) was popularized by Rumelhart et. al. [102], that deep NN started being used more widely. Yet, DNNs with a lot of layers suffered from an issue known as the vanishing or exploding gradient issue. This major issue of Deep

Learning refers to the exponentially plummeting or soaring of the value of error signals when they are propagated back through the network during BP. The more layers, the more prone the network was to suffer from this issue. Deep architectures rose to fame when new training methods were proposed [5, 7, 51], based on graph theory and on the use of a generative model called Restricted Boltzmann Machines (RBM) [33, 52], which subsequently became a tool of choice for pattern recognition tasks. More details about this model will be given in Chapter 3. In the speech community, deep architectures with generative pre-training have been shown to perform better than traditional (HMM-based) methods for automatic speech recognition [98].

Convolutional Neural Networks. Neural Networks have been adapted to image analysis in the form of Convolutional Neural Networks (CNNs), which can analyze 2-dimensional inputs. They were introduced in [34] as a visual pattern recognition technique unaffected by shifts in position. The general principle is that they can process 2-dimensional data by analyzing the input through small windows sliding over both dimensions of the whole input array. Each analysis window is composed of a 2D matrix of weights acting as a 2D-filter, hence the name 'convolutional'. A more indepth decription of the CNNs will be given in Chapter 3.

2.1.7 Sequence modelling with Neural Networks

A major aspect of the research on NNs in the speech/audio processing community is to study their ability to tackle time sequences, which is essential since, as we have seen in Section 2.1.2, sounds are defined through time, as an evolving process. In this subsection, we present the main approches that have been proposed to efficiently process temporal sequences with NNs.

Context-Dependent Neural Networks. One straightforward manner to adapt NNs to analyze temporal sequences is to adopt the context dependency principle described in Section 2.1.2: An input vector to a NN is formed by the concatenation of several single-frame input vectors. In the following, this approach is referred to as Context-Dependent Neural Networks (CD-NNs).

NN-HMMs. Another major early attempt to use NNs to process temporal sequences of vectors was made by Bourlard et al. [13] who constructed a hybrid of NNs and HMMs, in which the NNs were used to predict the frame-wise emission probabilities of the HMM states, along the same lines as GMM-HMMs [99].

Recurrent Neural Networks. Recurrent Neural Networks (RNNs) are a special kind of Neural Networks that was devised specifically for sequence modeling. RNNs are built to retain information from past inputs in a dedicated memory. This memory is contained in the hidden units (also often called 'cells' in the context of RNNs), who are able to retain and accumulate passed information and subsequently use it when necessary to output classification decisions. One basic manner to implement this principle is to set a recurring connection onto the neurons, as illustrated in Figure 2.12: At timestep n a neuron uses as inputs both the current outputs from previous layers (or the input data in case of first layer) and its own past output calculated at time-step $n-1$. This way, the information at a given time-step can be carried through several time-steps, enabling temporal integration of the information.

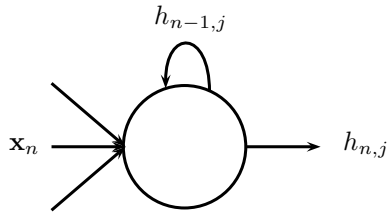

Figure 2.12: j^{th} recurrent neuron

Despite their natural ability to learn time structures, RNNs suffered from the vanishing/exploding gradient just like DNNs [8]. As a result, there were only a handful of successful applications of RNNs for speech and language modeling using large RNNs [81].

Long Short Term Memory. To overcome the gradient issues in RNNs, Hochreiter and Schmidhuber [53] came up in 1997 with a variant of RNNs, replacing the classic neurons (made of a unique elementary non-linear function) with (much) more elaborate cells called Long Short Term Memory (LSTM). LSTM cells are capable of encoding temporal information over a long period of time by using a separate 'container' variable, the state, where all the information is accumulated. This dual architecture, where memory from previous inputs is retained in a separate pipe, allows the main pipe to access that memory in order to make decisions taking into account information from a long-term past.

RNNs using LSTM cells were applied to handwriting recognition [46] and ASR [45]. LSTM networks will be described in more details in Chapter 3.

2.1.8 Post-processing for final decision

For NNs and RNNs, the estimation of class posterior probabilities is made for every frame-wise input vector $\mathbf{x}_n$, hence at every frame index from $n = 1$ to N. Depending on their implementation, this can also be true for CNNs and context-dependent NNs (if 1-frame shift is applied in the latter case). Depending on the ability of each frame of signal (or CD window, or frame + memory) to represent well the underlying class of sound, a more or less erratic sequence of outputs can be observed, switching from one class to another between two consecutive inputs, which generally makes poor sense since we have seen that sounds generally have a length that is significantly superior to a frame length. To address this issue, post-processing methods are used to filter the sequence of frame-wise outputs and provide a sequence of more consistent final "global" decisions at the sound length level. Two types of filtering methods are considered here:

- A smoothing algorithm,
- A majority voting algorithm.

Note that these methods can been seen as a very simplified version of the NN-HMM approach presented above, where the HMM part is seen as a filtering/smoothing process over the outputs of the NN.

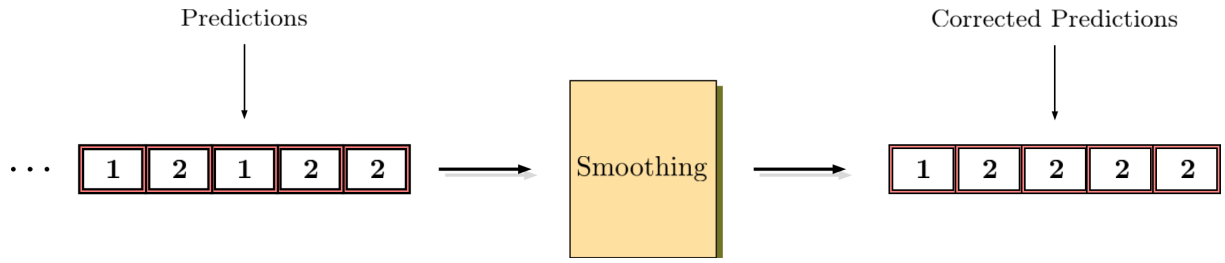

Figure 2.13: Output corrected by smoothing algorithm.

Smoothing algorithm. The first method prevents a decision at time n from diverging if the two directly adjacent decisions (i.e. at time $n-1$ and $n+1$) are the same, as shown in Figure 2.13. If the two side predictions are not identical, the algorithm does not do anything.

Majority voting algorithm. In the second method, the predictions over a segment of frames of a given length, denoted T_{seg}, are merged into one unique prediction for the whole segment. To this aim, a majority voting scheme is applied over the segment: The algorithm outputs the class which has the highest number of frame-wise decisions. As a result of this segment detection, the system now outputs one prediction every T_{seg} seconds (if no overlap exits between consecutive segments). Figure 2.14 gives an example.

2.1.9 Evaluation Metrics

To finish with the generalities of pattern recognition and its application to audio signals, we present the evaluation metrics that are used to measure the performance of the models as classifiers. To this aim, the following cases are defined, according to [79]:

- True Positive (TP): the system detects an event which is present in the input vector.
- True negative (TN): the system does not output an event which is not present in the input vector.
- False Positive (FP) (aka False Alarm): the system outputs an event which is not present in the input vector.
- False Negative (FN) (aka Missed Detection): the system does not output an event which is present in the input vector.

Based on those concepts, the number of occurrences of each case on a given test dataset are counted. Then the so-called Precision P and Recall R (or Accuracy) of the system are calculated as:

$$P = \frac{TP}{TP + FP}. \tag{2.15}$$

$$R = \frac{TP}{TP + FN}. \tag{2.16}$$

R represents the detection accuracy of the system. It shows how well the system was able to correctly detect the target events in proportion of the number of events to be detected. P is a

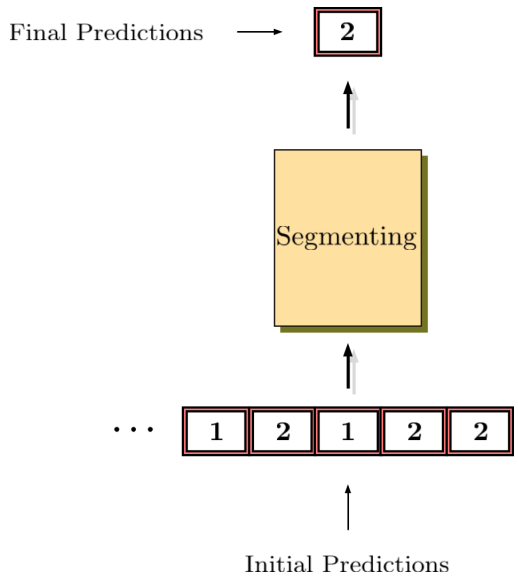

Figure 2.14: Detection by majority voting over segment of 5 consecutive frame-wise predictions.

measure of the detection quality but in proportion of the number of false alarms. Both values are essential to fully apprehend the system performance. Indeed, if R is high and P is low, the system was able to detect a large amount of the actual targets it was given, but also output a lot of false alarms. Conversely, if R is low and P is high, the system missed a lot of actual targets, but it did not cause a lot of false alarms.

Additionally, results are often reported in the form of the geometric mean between Precision and Recall, also called F-score:

$$F = \frac{2PR}{P+R}. \tag{2.17}$$

The results can also be reported in the form of a confusion matrix, where entry $e_{i,j}$ is the percentage of occurrences from class i classified as class j, which makes the diagonal elements of the confusion matrix correspond exactly to the Recall R. While FP and FN give an account of the amount of classification errors without giving any details about which classes were wrongly output, the advantage of the confusion matrix is that it breaks down the number of total errors (i.e. FP+FN) into the wrongly output classes, showing how the misclassified inputs were classified. For more clarity the highest numbers in each line of the confusion matrix are often displayed in bold, representing how the largest proportion of data from class i was classified.

2.2 Related works

Our field of application can be seen as part of a more general topic called Environmental Sound Recognition (ESR). ESR refers to the general task of detecting and classifying natural sounds from a continuous audio stream recorded by a microphone or a set of microphones placed in a natural environment. This problem can be further decomposed into Acoustic Scene Classification (ASC) and Sound Event Detection (SED) [3, 116]. In the following we present these two subtopics, then we provide examples of applications.

2.2.1 Environmental Sound Recognition

Acoustic Scene Classification. ASC deals with the recognition of the type of scene or environment in which the different successive or superimposed recorded sounds occur, e.g. office, restaurant, beach, street, forest, etc. [16, 20, 119]. As illustrated in the former examples, the environments can be indoor or outdoor [11]. The exact nature and time of occurrence of the different sounds composing the scene may not be a relevant target for this kind of problem. Following the principle of supervised training (see Section 2.1.1), for each type of scene/environment, a set of recordings taken from different instances of the considered environment is garnered. The environments instances are usually collected from different locations, in order to display variability. Indeed, the aim is generally to train a model to be able to recognize a given environment regardless of the location of the scene and the location of the microphone within that scene. An example of application is automatic robot orientation, where a robot needs to identify the type of environment in which it evolves [15]. Another type of application is to automatically identify the acoustic characteristics of the scene in order to adapt further specific sound processing, e.g. speech enhancement in noise.

Recognition accuracy varies greatly across studies, owing to the variety of the used datasets [15, 29, 70]. In [29] a GMM-HMM is applied on different features (MFCC, energy features, etc.) describing recordings from 24 acoustic scenes (restaurant, marketplace, café, etc.). In [70] MFCCs along with other spectral and energy features are used to describe similar acoustic scenes (15 classes taken from the DCASE challenge [116]). Again, similar features are used in [15] to describe 5 acoustic environments (café, lobby, elevators, outside and hallway). Because both target classes and experimental setups can be very different from one study to another, there is no clear way of evaluating the performances of a model for environment recognition. In 2016 however, a dataset was published to serve as a baseline for environment recognition studies [78]. This followed an effort to bring together the community through the annual DCASE 2016 challenge [116].

Sound Event Detection. As for SED, as explained in Section 2.1.1 the system is expected to detect the different elementary audio events occurring within a continuous audio stream, e.g. car engine whirring, glass shattering, footsteps, etc. Whenever possible, the system should provide their start and end times [118]. Therefore, in addition to the nature of the different classes (general type of acoustic scene vs. specific sounds), the difference between ASC and SED is rooted in the distinction between Classification and Detection outlined in 2.1.1. The CLEAR challenge in 2007 [110] attempted to provide a unified framework for the evaluation of SED methods, in which the metrics account for the system accuracy in determining the temporal boundaries of events. It has since been followed up by the DCASE challenge [116]. Obviously, the task addressed in the present thesis is part of the SED problem.

Even if each task has its own specificity, there is quite some overlap in the techniques used for ASC, SED, Music Information Retrieval (MIR) and Automatic Speech Recognition (ASR). In fact, ASR can be seen as a particular case of SED, but due to both huge applicative stakes and the specificity of speech signals, it was historically developed as a topic on its own. Common features can be used in all these tasks, including Mel-Frequency Cepstral Coefficients (MFCC), Mel-Spectrogram, signal statistics, Linear Prediction Coefficients (LPC) [50, 70]. Also, common models were considered, notably Gaussian Mixture Models (GMM), widely used in the early years of ASR, have been applied to scene recognition [3, 116]. Hidden Markov Models, which have been trending in ASR for decades, have been used for the recognition of specific sound events [77].

2.2.2 Applications

Robotics. As briefly stated above, classifiers for sound recognition have been used in Robotics to help understand the surrounding environment. An audition model is a prerequisite for natural human-robot interaction [57, 121]. Besides embedded ASR models [83, 87], it is useful for the robot to be able to recognize environmental sounds in order to understand the surrounding and scene context so as to better interact with people. Relatively short sounds with very distinctive starting and ending points (such as a door opening, a object dropping, etc..) are considered in [57] while [121] focuses on the detection of abnormal sound events such as glass breaking, gun shooting, screaming, etc..

Speech. Automatic speech recognition is a research topic that has spanned decades. Classifiers are used to recognize sub-word entities, usually defined as phonemes or syllables, in a framework called Acoustic Modeling. Another layer of classification is used to perform what is referred to as the Langage Model. These two models are combined together to perform speech recognition. The most popular technique used in this field for decades has been the Hidden Markov Model (HMM) [2, 60, 73, 99, 124]. In 2015, state-of-the-art unconstrained ASR (no constraint on speaker identity, presence of low-to-notable environmental noise) consists of a combination of very deep DNNs with CNNs and LSTMs trained on huge speech databases (tens of thousands of hours of speech material) [104].

Audio surveillance. This topic of application will be of particular interest to us since it is closely related to the final objective defined for this thesis, namely the detection of screams and shouts. Therefore it appears natural that it should be described in more details than the previous fields of application covered so far. The detection of critical situations is a highly coveted topic in today's security-related considerations. It is found in numerous distinct research areas such as Computer Vision, Intelligent Surveillance and Audio Event Detection. While most Detection systems are based on visual information [93, 107, 123], the detection of abnormal situations based on audio cues has also been investigated [18, 61], and audio surveillance is now a trending field of application for sound recognition technologies. It requires solid performances from the model as it ideally aims at detecting all targeted sounds while keeping the false alarms rate as low as possible. In [19], two models are built on training data: one for gunshot sounds and another one for 'normal sounds (all other sounds). A decision is taken every 50 ms by selecting the class according to the model with the highest posterior probability across that window. In a second step, three models are built to describe more accurately the gunshot class, which is decomposed in three sub-classes. The classification decision is taken in three stages; the posterior probability of the model for the

normal class is compared to each one of the three models for the gunshot classes, and the data is classified as shot if at least one of the gunshot models yields higher posterior probability than the model for the normal class. The features used are MFCC coefficients and spectral statistical moments, along with their first and second derivative, on which Principal Component Analysis is performed. The data is built synthetically by mixing clean gunshot sounds with noise sounds with various Signal to Noise Ratio (SNR). [38] classifies both gunshot and screams. Some novel features are devised, based on the auto-correlation function in order to describe the energy over different time lags and thus differentiate screams, where the energy is rather spread across the duration of the event, and gunshot where the energy is contained in the first time lags. A feature selection process is used to reduce the dimensionality of the data. Two GMMs are used in parallel to discriminate between screams and noise on the one hand, and between gunshot and noise on the other hand. Again, as in [19], the data is artificially built by mixing clean recording of screams and gunshot sounds with noise according to different SNR ratios. [95] aims at detecting shouts in noisy conditions, although the noise is simulated here. MFCC coefficients are used as features and a frame selection process selects the highest energy frame. A GMM is then used to model the selected frames. Three classification rules are used: the first two consider shout versus non-shout but use two different likelihood functions, a two-stage rule separating speech from the rest first, then shout vs non-shout.

Security in public places and transportation. [101] presents a system which detects dangerous situations by detecting shouts in an audio stream captured in a real-life environment. A front-end performs segmentation of the stream via the Forward-Backward Divergence algorithm. Each segment is modelled by an auto-regressive models and a transition between two segments is detected when a change in the auto-regressive model is detected. After having discarded irrelevant segments this way, features are extracted from the raw audio in several ways: MFCC, LPC and PLP. A Gaussian Mixture Model (GMM) and a Support Vector Machine (SVM) are then applied to classify the segments. A hierarchical decision tree is built with a binary classifier at each step; the first one aims at discarding short-term noises, the second one, taking as input only the segments not classified as short-term noises by the first one, performs a speech/non-speech classification, feeding the third one with segments classified as speech so that the final decision can be made, classifying into shout/non-shout. The front-end and hierarchical decision tree are deemed useful by performing A/B comparison on training data. Along the same lines, [113] presents a system to detect screams and gunshots, which employs spectral features as well as features based on the autocorrelation function, on top of the classic MFCC coefficients. A feature selection process singles out the features which minimize an objective function. Two GMM classifiers running in parallel discriminate screams and gunshots from noise. As in [101], the input data is recorder in real-life conditions.

2.2.3 Polyphonic Sound Event Detection

Most SED tasks dealing with real-world applications are confronted with the practical issue of polyphonic audio. Indeed, the acoustic environment generally comprises multiple sources occurring at any given time, which means that sound events can (and often do) overlap. Synthetically created datasets attempt to recreate this polyphony [19, 62] while recordings of live situations are simply naturally subject to it [64]. When single-label classifiers are used, as introduced in Section 2.1.3, they are said to be monophonic in the audio framework, because they can only recognize

one sound event/source at a time. If those classifiers are applied as is to polyphonic audio data, a monophonic labelling scheme is obtained, inevitably causing some loss of information. Indeed, since only one label can be present at a given time, one would have to disregard other potential events occurring at that same time. Often, such labelling methods are based on the most prominent event [77]. The multi-label paradigm defined in Section 2.1.3 can be used to better process polyphonic audio. As an example, [90] used a multi-label RNN for environmental sound recognition, with sigmoid output activation function and a cross-entropy likelihood function for learning.

2.3 Conclusion

In this chapter we described the topic of pattern recognition and showed how it is relevant to our problem. More specifically we introduced Classification and explained how it can be used in the context of detection of audio events. The issue of dealing with sequential data was raised and the main difficulties linked to it were exposed, namely the segmentation of data and the variable lengths of patterns. These two aspects must be taken under consideration in our study since as we mentioned audio patterns are sequences with temporal signature. In this context, Context Dependency provides a way to segment the audio data while keeping the temporal information, and pre-segmented and streaming are two approaches providing different ways to go about the problem of the variable length of patterns. The former one ignores the streaming nature of the audio data and focuses on the classification of individual elements, whereas the latter takes it into account. We introduced the multi-label classification paradigm which is useful when several independent audio sources are present in the environment.

The mathematical models used to solve all these questions were described, starting with generative models (GMM, HMM) then discriminative ones (SVM, ANN). More details about the way sequences can be handled by ANNs were given, as they constitute the centerpiece of the work presented in the following. Specifically, the RNN, a model designed to provide a way to properly handle sequences, is presented along with its extension with LSTM units. Those units are made to overcome a major training issue of classical RNNs. Two post-processing methods were introduced, which will be used to improve the model's predictions. Finally, we explained how the models will be tested in the following and how their performance will be tested.

The second part of this chapter was dedicated to the presentation of works and studies relying on classification methods, which have been conducted in different fields of application.

Chapter 3

Theory of Neural Networks

The problem of sound event detection presented in the introduction is tackled from a classification perspective, as introduced in Section 2.1.1, using Artificial Neural Networks (ANNs), introduced in Section 3.1. Only the basics of ANNs were provided in Section 3.1. The purpose of the present chapter is thus to provide the reader with more details on the mathematical formalism derived from the theory of Neural Networks. chapter can shed some light on certain areas which may still be unclear. Most mathematical developments presented in this chapter are directly inspired from [10].

3.1 From linear models to Deep Neural Networks

To progressively introduce and justify the formalization of ANNs equations, we start from a type of relatively 'simple' Discriminative models called Linear Models and we explain how they are relevant to Classification. Then we will go from Linear Models to ANNs, following the general line of [10].

3.1.1 Linear Models for Classification

Being discriminative, Linear Models seek to estimate the posterior probability of classes $p(c \mid \mathbf{x})$. Their output is defined as a monotone function f of a linear combination between input $\mathbf{x}$ and a weight vector $\mathbf{w}_c$ defined for each class c:

$$y_c(\mathbf{x}) = f(\mathbf{w}_c^T \mathbf{x} + w_{c0}), \tag{3.1}$$

where $y_c(\mathbf{x})$ is the output of the model for class c, $(.)^T$ is the transpose operator, and w_{c0} is a bias term. The set of parameters θ here consists of all weight vectors $\mathbf{w}_c$ and all biases w_{c0}. As will appear clearer in the following, f is generally a non-linear (monotone) function, so the name of the model is to be seen as referring to the form of the input to this function.

Now we will see under which condition(s) Equation (3.1) is equivalent to $p(c \mid \mathbf{x})$. Recall that:

$$\begin{aligned} p(c \mid \mathbf{x}) &= \frac{p(\mathbf{x} \mid c)p(c)}{p(\mathbf{x})} \\ &= \frac{p(\mathbf{x} \mid c)p(c)}{\sum_{i=1}^{C} p(\mathbf{x} \mid i)p(i)}, \end{aligned} \tag{3.2}$$

with C being the total number of classes. If we define a_c as:

$$a_c = \log(p(\mathbf{x} \mid c)p(c)), \tag{3.3}$$

we get, for the posterior of class c :

$$\begin{aligned} p(c \mid \mathbf{x}) &= \frac{exp(a_c)}{\sum_{i=1}^{C} exp(a_i)} \\ &= Softmax(a_c), \end{aligned} \tag{3.4}$$

which is the equation of the Softmax function. In a two-class problem (where $C = 2$) this expression simplifies to:

$$\begin{aligned} p(c = 1 \mid \mathbf{x}) &= \frac{p(\mathbf{x} \mid c = 1)p(c = 1)}{p(\mathbf{x} \mid c = 1)p(c = 1) + p(\mathbf{x} \mid c = 2)p(c = 2)} \\ &= \frac{exp(a_1)}{exp(a_1) + exp(a_2)} \\ &= \frac{1}{1 + exp(-a)} \\ &= \sigma(a), \end{aligned} \tag{3.5}$$

where $\sigma(a)$ is the sigmoid function and a is defined as:

$$a = \log \frac{p(\mathbf{x} \mid c = 1)p(c = 1)}{p(\mathbf{x} \mid c = 2)p(c = 2)} = a_1 - a_2.$$

Now, if we make the assumption that each class conditional probability is a Gaussian, with mean μ_c, and same covariance matrix $\mathbf{\Sigma}$ for all classes, we have:

$$p(\mathbf{x} \mid c) = \frac{1}{(2\pi)^{D/2} \mid \Sigma \mid^{1/2}} \exp(-\frac{1}{2}(\mathbf{x} - \mu_c)^T \Sigma^{-1}(\mathbf{x} - \mu_c)), \tag{3.6}$$

and a_c can be redefined as:

$$a_c = -\frac{1}{2}(\mathbf{x} - \mu_c)^T \mathbf{\Sigma}^{-1}(\mathbf{x} - \mu_c) + \log p(c). \tag{3.7}$$

From Equation (3.4) it follows that the ratio of the log-posterior of each pair of two classes i and j becomes:

$$\begin{aligned} \log \frac{p(c = i \mid \mathbf{x})}{p(c = j \mid \mathbf{x})} &= a_i - a_j \\ &= \mathbf{w}_{ij}^T \mathbf{x} + w_{ij0}, \end{aligned} \tag{3.8}$$

with

$$\mathbf{w}_{ij} = \mathbf{\Sigma}^{-1}(\mu_i - \mu_j) \quad \text{and} \quad w_{ij0} = -\frac{1}{2}\mu_i^T \mathbf{\Sigma}^{-1}\mu_i + \frac{1}{2}\mu_j^T \mathbf{\Sigma}^{-1}\mu_j + \log \frac{p(c = i)}{p(c = j)}. \tag{3.9}$$

This ratio represents the boundary between classes i and j since it defines the equation of a hyperplane where those classes are equiprobable.

Now, from (3.4) and (3.7), we can deduce the posterior probabilities for each class c:

$$p(c \mid \mathbf{x}) = \frac{exp\big(-\frac{1}{2}(\mathbf{x} - \mu_c)^T \Sigma^{-1}(\mathbf{x} - \mu_c) + \log p(c)\big)}{\sum_{i=1}^{C} exp\big(-\frac{1}{2}(\mathbf{x} - \mu_i)^T \Sigma^{-1}(\mathbf{x} - \mu_i) + \log p(c = i)\big)}. \tag{3.10}$$

The quadratic terms in $\mathbf{x}$ are irrelevant because they appear as a constant quantity in each posterior probability. Therefore they can be removed, and keeping only the rest leads to the logistic regression equation:

$$p(c \mid \mathbf{x}) = softmax(\mathbf{w}_c^T \mathbf{x} + w_{c0}), \tag{3.11}$$

with

$$\mathbf{w}_c = \mathbf{\Sigma}^{-1}\mu_c \quad \text{and} \quad w_{c0} = -\frac{1}{2}\mu_c^T \mathbf{\Sigma}^{-1}\mu_c + \log p(c). \tag{3.12}$$

Equation (3.11) is exactly the expression of Equation (3.1) if f is chosen to be the Softmax function. For the 2-class problem we have a similar identification using the sigmoid function. This reasoning holds for any distribution of the exponential family, not just Gaussian distributions.

To summarize, assuming Gaussianity for the class conditional distributions with identical covariance matrix for all classes, in a two-class problem the sigmoid function is used to compute the posterior probabilities of class as in Equation (3.5), while in a multi-class problem the Softmax function is used as in Equation (3.4).

Basis Functions

Although the models we have discussed so far simply take the input as it is, in some cases the classes are not linearly separable, meaning that the equation of the equiprobability hyperplane separating the classes is no longer a linear combination of the input, but a more complex function. In other words, a simple linear combination of the input can no more efficiently characterize each class. To remedy this, some models operate a non-linear transformation of the input into feature vectors $\phi(\mathbf{x})$, as was already examplified with the SVM model, in Section 2.1.5. The idea is to find a transformation which would make the classes linearly separable in the feature space. In the present context, the function ϕ is referred to as a basis function, and the model is no longer linear in the inputs but is linear in the feature vectors $\phi(\mathbf{x})$:

$$y_c(\mathbf{x}) = f(\mathbf{w}_c^T \phi(\mathbf{x}) + w_{c0}). \tag{3.13}$$

Because they keep the notion of linear combination of some input elements, the theory derived for linear models can still hold as long as the features are assumed to follow an Exponential distribution.

Logistic Regression

The Logistic Regression model, which despite its misleading name is in fact a classification model, is a discriminative model used for binary classification. It chooses out of two classes c_1 and c_2 by performing a non linear transformation of a linear combination of the feature vector $\phi(\mathbf{x})$ with a weight vector $\mathbf{w}$, as described in the previous paragraph:

$$y(\mathbf{x}) = f(\mathbf{w^T} \phi(\mathbf{x}) + w_0). \tag{3.14}$$

Its name refers to the use of the sigmoid function used as the transformation function to make predictions for the posterior probability $p(c \mid \mathbf{x})$, which fall within the range $[0, 1]$. The following test is generally used to determine the target class:

- $c = c_1$ if $p(c \mid \mathbf{x}) \leq T$
- $c = c_2$ if $p(c \mid \mathbf{x}) > T$,

where T is a threshold (usually set to 0.5).

As for the linear models above, several logistic regression units can be used to perform classification tasks where more than two classes are at stake (i.e the multi-class problem). In that case, each unit is associated with one class c and uses the Softmax function as explained in Section 3.1.1.

Optimization / Learning via Stochastic Gradient Descent

Here, we provide the basics of the optimization (or learning) of the Logistic Regression. This will form the basis of the training principle of ANNs, as explained later.

Target vector. To this aim, we need to formalize the concept of class c in mathematical form, and we define the so-called target (or label) vector that represents the true class of the input vectors. This target vector is denoted $\mathbf{t}$ and is of size C (the number of classes); each of its entry represents a class c and is set to 1 if the vector belongs to it, and 0 otherwise. For example, in a 3-class problem where the model is shown some animal pictures and must decide which animal is represented out of {Cat, Dog, Horse}, the target vector corresponding to a dog picture would be $\mathbf{t} = [0, 1, 0]^T$.

Likelihood function. Here we consider supervised learning: we suppose that each data vector $\mathbf{x}_n$ of the training database is annotated, i.e it is associated with a target vector $\mathbf{t}_n$. describing its corresponding class. Optimization of the model (i.e. tuning of the parameters) described here is based on the maximization of a likelihood function defined on the training dataset with respect to the complete set of parameters, denoted $\mathbf{W}$ (this complete set of parameters is composed of all weight vectors and all biaises of the model). The fundamental idea is to adjust the weights of the model to make the labels more likely given their corresponding input. With the assumption that the data pairs $\{\mathbf{x}_n, \mathbf{t}_n\}$ are independent one from each other, the likelihood function of the entire training dataset is equal to the product of the likelihood function evaluated for each training data pair. For simplicity of presentation, in the following we present the likelihood function (and the corresponding optimization equations below) considering only one data pair $\{\mathbf{x}, \mathbf{t}\}$. These equations generalize to the likelihood function evaluated on the complete training dataset. Formally, the likelihood function is defined as the conditional distribution of target vector $\mathbf{t}$, given the corresponding input and the model parameters[1]:

$$L(\mathbf{W}) = p(\mathbf{t}|\mathbf{x}, \mathbf{W}). \tag{3.15}$$

To further derive this expression, several scenarios can be identified depending on the final task, with different types of relations between the units. For binary classification there is only one unit; for multi-class single-label classification there are C units which are mutually exclusive (only one is valued 1, all others are zero); and for multi-class multi-label classification (as defined in Section 2.1.3), there are C units which are independent. 2-class binary classification involves a single unit and a single scalar target variable t taking values 1 and 0 for class c_1 and c_2 respectively. We have:

$$y(\mathbf{x}, \mathbf{W}) = p(t = 1|\mathbf{x}, \mathbf{W}) = p(c_1 \mid \mathbf{x}), \tag{3.16}$$

$$p(t = 0|\mathbf{x}, \mathbf{W}) = p(c_2 \mid \mathbf{x}) = 1 - y(\mathbf{x}, \mathbf{W}). \tag{3.17}$$

Therefore, the variable t represents the outcome of a binary experiment with probabilities $y(\mathbf{x}, \mathbf{W})$ and $1 - y(\mathbf{x}, \mathbf{W})$, thus following a Bernouilli distribution with parameter $y(\mathbf{x}, \mathbf{W})$. Its conditional distribution, i.e. the likelihood function, can then be written as

$$p(t|\mathbf{x}, \mathbf{W}) = y_c(\mathbf{x}, \mathbf{W})^t (1 - y_c(\mathbf{x}, \mathbf{W}))^{(1-t)}. \tag{3.18}$$

[1]Hence the likelihood function is presented as a function of the target vector, but when used for training it must be considered as a function of the parameter set, given fixed values of the training data.

In a multi-class single-label paradigm (with $C > 2$) the neurons have Softmax activation, as seen in Section 3.1.1. They act as one unique multi-dimensional unit instead of multiple independent units, because the Softmax activation function forces the output probabilities to sum up to one. Therefore, the label vector is a C-dimensional random variable, and its conditional distribution is given by the following categorical distribution[2]:

$$\begin{aligned} p(\mathbf{t} \mid \mathbf{x}, \mathbf{W}) &= \prod_{c=1}^{C} y_c(\mathbf{x}, \mathbf{w}_c)^{t_c} \\ &= \prod_{c=1}^{C} \Big(\frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{i=1}^{C} \exp(\mathbf{w}_i^T \mathbf{x})} \Big)^{t_c}. \end{aligned} \tag{3.19}$$

In the multi-class multi-label case (i.e the input vector can belong to several classes at the same time, as described in Section 2.1.3), we can use C units with sigmoid activation functions, to perform C binary classifications. Each unit compute the posterior probability of its corresponding class c independently of the results from other units. Therefore the units can be considered independent and the target vector made of independent random variables, each one following a Bernouilli distribution. As a result, the conditional distribution of the target vector is here the product of the probability of each of its entries:

$$p(\mathbf{t} \mid \mathbf{x}, \mathbf{W}) = \prod_{c=1}^{C} p(t_c \mid \mathbf{x}, \mathbf{w}_c) = \prod_{c=1}^{C} y_c(\mathbf{x}, \mathbf{w}_c)^{t_c} (1 - y_c(\mathbf{x}, \mathbf{w}_c))^{(1-t_c)}. \tag{3.20}$$

Optimization. Whatever the form of the likelihood function, maximization of that function is generally performed by minimizing its negative log value, which is easier to manipulate (both mathematically and computationally). This is done via an efficient iterative algorithm called Stochastic Gradient Descent (SGD), a stochastic approximation of the Gradient Descent method for finding the minimum of a function. According to this method, the model parameters $\mathbf{W}$ should be updated at every iteration by the following formula:

$$\mathbf{W}^{i+1} = \mathbf{W}^i - \Delta E(\mathbf{W}^i), \tag{3.21}$$

where $E(\mathbf{W})$ is the cost function to be minimized, i.e the negative log-likelihood here, Δ is the derivation (gradient) operator, and i is the iteration index. To perform the SGD update of Equation (3.21), the negative log-likelihood function is evaluated on the entire training dataset $\{\mathbf{x}_n, \mathbf{t}_n\}_{n=1}^N$ or on subsets of this training set (this principle be further described in the experiments reported in the later chapters). Again, for simplicity of presentation, we only consider one input/output data pair.

In the multi-class single-label case, from Equation (3.19), the negative log-likelihood writes:

$$\begin{aligned} E(\mathbf{W}) &= -\log(p(\mathbf{t} \mid \mathbf{x}, \mathbf{W})) \\ &= -\sum_{c=1}^{C} t_c \log \Big(\frac{\exp(\mathbf{w}_c^T \mathbf{x})}{\sum_{i=1}^{C} \exp(\mathbf{w}_i^T \mathbf{x})} \Big) \\ &= -\sum_{c=1}^{C} t_c \Big(\mathbf{w}_c^T \mathbf{x} - \log \big(\sum_{i=1}^{C} \exp(\mathbf{w}_i^T \mathbf{x}) \big) \Big). \end{aligned} \tag{3.22}$$

[2]Here, for simplicity of presentation, the bias term w_{c0} is included in the vector $\mathbf{w}_c$ and a corresponding dummy variable 1 is added into the input data vector $\mathbf{x}$.

Thus, the derivative of the cost function $E(\mathbf{W})$ with respect to weights $\mathbf{w}_c$ associated with class c is simply calculated as follows:

$$\begin{aligned}\Delta E(\mathbf{w}_c) &= -t_c\mathbf{x} + t_c\frac{\mathbf{x}\exp(\mathbf{w}_c^T\mathbf{x})}{\sum_{i=1}^{C}\exp(\mathbf{w}_i^T\mathbf{x})} \\ &= y_c(\mathbf{x},\mathbf{w}_c) - t_c.\end{aligned} \tag{3.23}$$

Gradient derivation gives similar results in the binary case and in the multi-class multi-label case. Moreover the weight update is done independently for each unit, with its own error and its own weight matrix. This result is in fact the same for every pairing of cost function and output activation function.

3.1.2 The Multi-Layer Perceptron and (Deep) Neural Networks

The general architecture of Artificial Neural Networks has already been given in Section 3.1. After inspecting the Logistic Regression, we may say that the basic idea of (direct feedforward) ANNs is to connect together a set of Logistic Regression units in both parallel, within layers, and in series (successive layers), in order to build more general classifiers of arbitrary complexity. The resulting model, historically called Multi Layer Perceptron (or MLP) then later NNs and deep NNs, is organized in L hidden layers, each of J_l Logistic Regression units, also referred to as 'neurons' or 'cells', as described in Section 3.1. A NN with two hidden layers is illlustrated in Figure 3.1 (already displayed in Chapter 2), here the two hidden layers have the same number of units, but this is not mandatory). Each neuron j in layer l acts as a Logistic Regression, transforming its input according to Equation (3.14) to compute its output $h_j^{(l)}$:

$$h_j^{(l)} = f(\mathbf{w}_j^{(l)T}\mathbf{h}^{(l-1)} + w_{j0}^{(l)}) \tag{3.24}$$

where $\mathbf{h}^{(l-1)}$ is the output vector of layer $(l-1)$. For the first hidden layer ($l=1$), we have: $\mathbf{h}^{(0)} = \mathbf{x}$. For the output layer, each neuron computes:

$$y_c = g(\mathbf{w}_c^T\mathbf{h}^{(L)} + w_{c0}), \tag{3.25}$$

where c is the output neuron index, corresponding to the associated class, L is the number of hidden layers, and g is the Softmax function so that the NN output is consistent with class posterior probability values, as seen in the previous subsection (for a 2-class problem, g is the sigmoid function):

$$y_c = p(c \mid \mathbf{x}). \tag{3.26}$$

The non-linearity stemming from function ϕ as defined in Equation (3.14) arises here from the succession across layers of linear combinations of the successive inputs with weights and non-linear activation functions, as described by Equation (3.24).

Learning (D)NNs. To learn (D)NNss an efficient algorithm was proposed and analyzed in [102]. This algorithm, which is very famous in the NN and deep learning community, is generally referred to as the 'error backpropagation algorithm'. We try to present rapidly the principle and

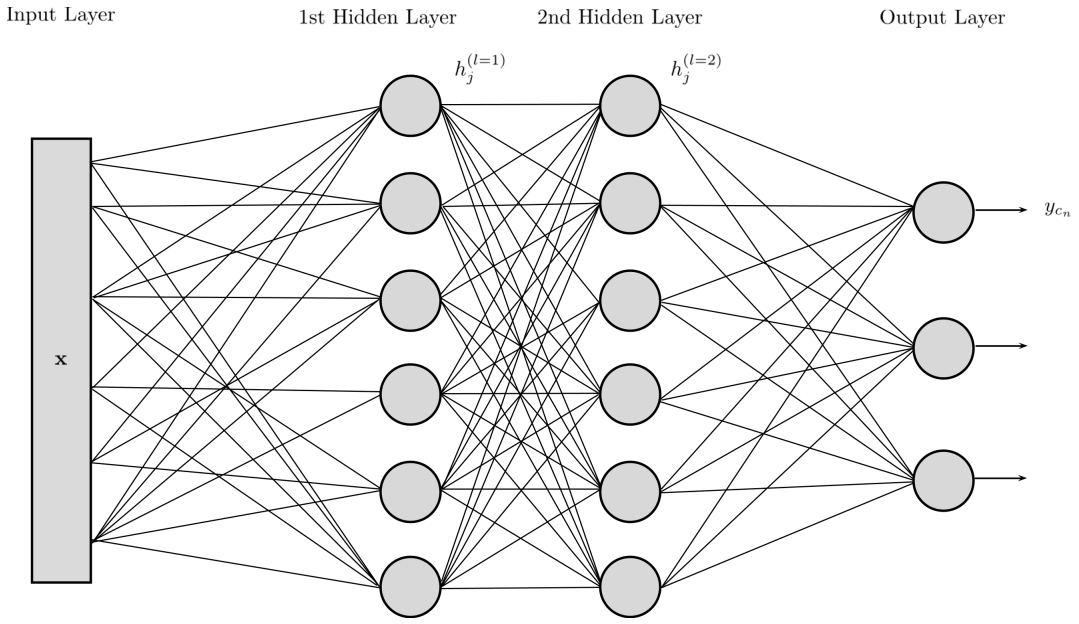

Figure 3.1: An example of Neural Network with two hidden layers.

foundations of this algorithm here, without entering into too much detail. To make the presentation simpler we re-write Equation (3.24) as:

$$h_j^{(l)} = f(a_j^{(l)}) \qquad \text{with} \quad a_j^{(l)} = \sum_i^I w_{ij}^{(l)} h_i^{(l-1)} + w_{j0}^{(l)}. \tag{3.27}$$

(D)NNs are generally used for multi-class single-label tasks, so we consider here the likelihood function defined by Equation (3.19). As before, the SGD method is used to adjust the network weights to the training data. Therefore, the gradient of this likelihood function needs to be evaluated with respect to each and every weight in the network $w_{ij}^{(l)}$ connecting unit i of layer $l-1$ to unit j of layer l. For readability in the derivations below, we write the cost function E intentionally omitting variable $\mathbf{W}$, which E depends on. The weight $w_{ij}^{(l)}$ has a contribution to the cost function E through Equation (3.27), thus the gradient can be expressed with the chain rule of derivatives:

$$\frac{\delta E}{\delta w_{ij}^{(l)}} = \frac{\delta E}{\delta a_j^{(l)}} \frac{\delta a_j^{(l)}}{\delta w_{ij}^{(l)}} = \frac{\delta E}{\delta a_j^{(l)}} h_i^{(l-1)}, \tag{3.28}$$

with $h_i^{(l-1)}$ the output of unit i from layer $l-1$ which is connected to unit j in layer l through the weight $w_{ij}^{(l)}$. This shows that $\frac{\delta E}{\delta w_{ij}^{(l)}}$ is obtained by multiplying the gradient of the cost function with respect to $a_j^{(l)}$ by the output of unit i from layer $l-1$. To compute that 'new' gradient $\frac{\delta E}{\delta a_j^{(l)}}$, applying the chain rule of derivative again shows that it involves a sum over the units located in higher layers ($l+1$, $l+2$, etc.) of the same terms from those layers ($\frac{\delta E}{\delta a_k^{(l+1)}}$, $\frac{\delta E}{\delta a_k^{(l+2)}}$, etc.). Therefore Equation (3.28) is repeated in those layers, all the way to the last layer. Now, at the last layer,

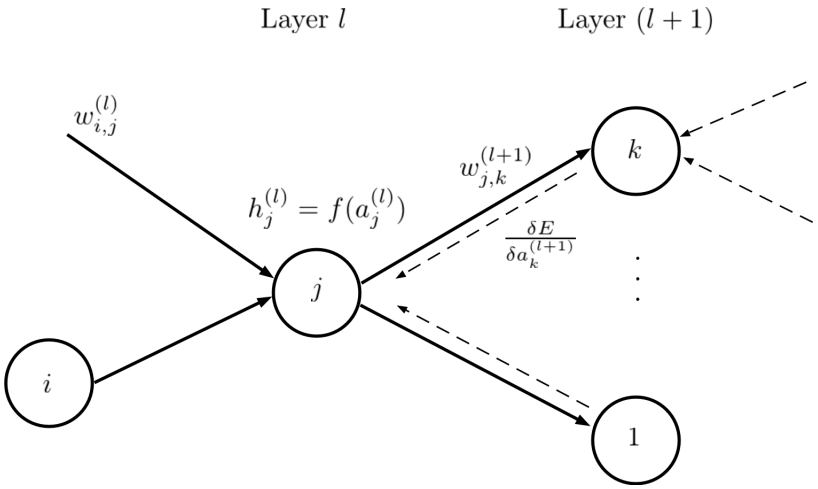

Figure 3.2: Illustration of Back-Propagation of errors for neuron j in layer l.

the derivative of the error with respect to the softmax units' inputs a_j is derived from the logistic regression case, as seen in Equation (3.23):

$$\frac{\delta E}{\delta a_j^{(l)}} = y_c(\mathbf{x}, \mathbf{w}_c) - t_c \tag{3.29}$$

So computing this term at the output layer, we can then compute the $\frac{\delta E}{\delta a_j^{(l)}}$ in all hidden layer one by one by using the error derivative from higher layers (per Equation (3.28)), as shown in Figure 3.2. This justifies the name of the algorithm: in short, the derivative of the error function defined in Equation (3.23) is propagated from the last layer back into lower layers.

3.1.3 Generative pre-training

Introduction

One issue with NNs is that the cost function may be non-convex and may display several local minimas, in which the parameters might get stuck during training, depending on their initial value. This is why parameters initialization is crucial. Initial values resulting in a lesser initial cost that is closer to the global minimum will allow gradient descent to avoid some local minimas and reach the global minimum more easily. To achieve this, it was proposed in [51] to train NNs with a generative model first: the Restricted Boltzman Machine (RBM). Generative models, as presented in Section 2.1.4, are designed to model the statistical process that spawned the observed data along with the missing information sought to be estimated. When applied to a structure involving hidden variables, such models learn the interactions between visible ($\mathbf{x}$) and hidden ($\mathbf{h}$) variables. This is useful for Classification as it can provide additional knowledge about the internal structure of classes.

Restricted Boltzmann Machines

The Restricted Boltzmann Machine is based on a type of probabilistic models, called Graphical Models, which describe the interactions between random variables in a network through certain properties. RBMs consist of one layer of visible variables $\mathbf{x}$, and another layer of H hidden variables $\mathbf{h}$, where each variable is connected to all variables from the other layer, and to none from its own layer, exactly as the neurons in a Neural Network, see Figure 3.3. RBMs rely on the assumption that hidden units are just a part of a bigger stochastic process which includes the training dataset. As such, the hidden units represent variables belonging to the same process as the training data, but which are missing from the dataset. During the training phase, RBMs learns the links and interactions between variables in the hidden and visible layers. This way they can provide a strong insight into possible connections between variables from two consecutive layers in a NN. This is why RBMs can be used to initialize weights in NN. Since RBMs are generative models and therefore estimate the distribution of the process encompassing both hidden and visible variables $p(\mathbf{x}, \mathbf{h})$, it is possible to find values of the weights which maximizes this joint probability. This is what is done during the learning process. Note that RBMs can be trained on unlabelled data since they do not require target values, as can be seen in the following derivations.

Formally, as for NNs, RBMs embed the interactions between variables through a set of weights $\mathbf{W}$ connecting the variables together. These interactions are embodied by a quantity called the energy defined by:

$$energy(\mathbf{x}, \mathbf{h}) = -\mathbf{b}^T\mathbf{x} - \mathbf{c}^T\mathbf{h} - \mathbf{h}^T\mathbf{W}\mathbf{x}, \tag{3.30}$$

where $\mathbf{b}$ and $\mathbf{c}$ are vectors including all the individual weights b_i and c_j associated to individual units. The corresponding probabilistic model of observed and hidden variables within a RBM is

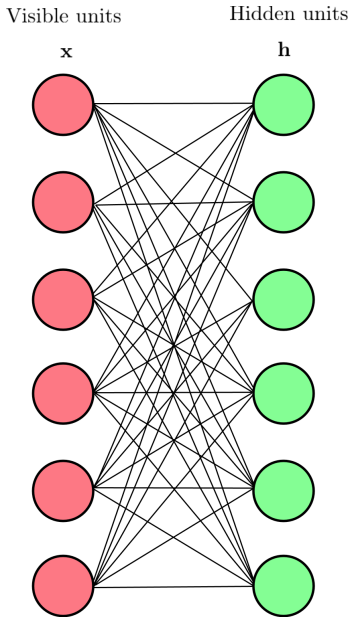

Figure 3.3: Connection between visible and hidden variables in a RBM.

defined by:

$$p(\mathbf{x}, \mathbf{h}) = \frac{e^{-energy(\mathbf{x},\mathbf{h})}}{Z} \tag{3.31}$$

where Z is a normalizing constant.

RBMs have the particularity to boast certain interesting properties:

- Each visible variable is independent from other visible variables given the hidden variables.
- Each hidden variable is independent from other hidden variables given visible variables.

Therefore:

- The posterior distribution of visible variables factorizes over hidden variables.
- The posterior distribution of hidden variables factorizes over visible variables.

The main equation stemming from these properties, which is a key in the learning process of RBMs, is the following:

$$p(\mathbf{h} \mid \mathbf{x}) = \prod_{j=1}^{H} p(h_j \mid \mathbf{x}), \tag{3.32}$$

where $p(h_j \mid \mathbf{x})$ is the posterior probability of individual hidden variable h_j and H is the total number of hidden variables. Because of the form of this posterior distribution of hidden variables (being a product over individual hidden variables), it can be computed easily, which makes gradient computation feasible (to perform negative log-likelihood minimization for weight omptimization) through the following equation:

$$\frac{\partial \log p(\mathbf{x})}{\partial \mathbf{w}_{ij}} = \langle \mathbf{x}_j \mathbf{h}_j \rangle_{data} - \langle \mathbf{x}\mathbf{h} \rangle_{model} \tag{3.33}$$

where the term $\langle vh \rangle_{model}$ is the expectation under the model's distribution. For more details about the theory of Graphical Models and RBMs, see **??**.

Fine-tuning

As with discriminative models, it is possible to create deep architectures of generative models in order to capture hidden features from the data, similarly to the way NNs are created. For instance, in [51] multiple RBMs were stacked onto each other to form a new architecture called Deep Belief Network (DBN), as illustrated in Figure 3.4, and an efficient training algorithm was proposed for such an architecture. Each RBM is trained one by one, starting from the bottom one. After training one layer on the training data, the next layer is trained using the hidden variables of that layer as inputs, and so on. After this training stage, the deep architecture was then 'converted' into a NN by only adding one layer with Softmax units on top of it to output classification predictions. This way, the resulting NN is nicely initialized with weights corresponding to a feature representation of the data. Further training this NN with a discriminative classification criterion as seen previously, forces the weights to adapt their feature extraction power to the classification task. This last phase is called fine-tuning.

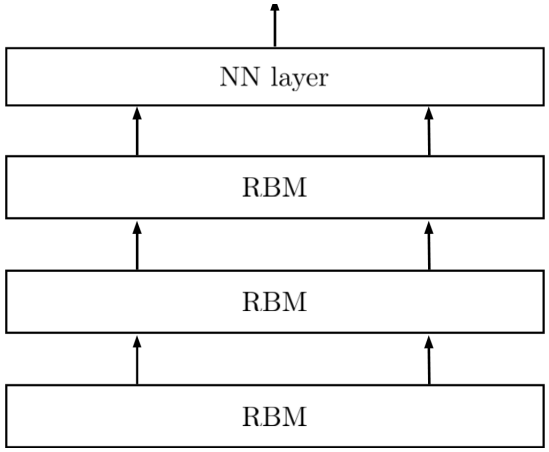

Figure 3.4: A DBN formed with stacked RBMs.

3.2 Convolutional Neural Networks

While NNs can be used on data vectors spanning a certain length across a second dimension via concatenation of consecutive input vectors into bigger input vectors, as explained in Section 2.1.2, they are inherently bound to consider the resulting input as a 1-dimensional vector. Although they might be able to detect and identify some patterns stemming from the second dimension of the input vector, any modulation of the patterns across that dimension would hamper this ability. This is because NNs are simply not designed for 2-dimensional patterns. In contrast, as briefly mentioned in Section 2.1.6, Convolutional Neural Networks (CNNs) were designed to handle 2-dimensional patterns, with the property that the analysis is unaffected by shifts of the patterns in position. This is possible via two powerful features. The first feature is the use of a series of 2D weight matrices acting as 2D sliding filters over the 2D input. Both data dimensions are then analyzed jointly in order to detect patterns across each axis. The physical meaning of each axis varies according to the data considered and the application of course. In image recognition for instance they would represent both dimensions of the images [66]. In our case, one of the axis may represent the local spectral content of sounds and the second axis may represent the time dimension, enabling the CNN to detect temporal patterns (in short spectrograms can be considered as images). Following that spirit, CNNs have been applied to speech processing [1]. The second feature making CNNs suitable for 2-dimensional inputs is the use of a max-pooling operation, explained below, which makes them robust to modulation. This way a given pattern can be detected regardless of the different length across either dimensions of distinct instances of that pattern.

The different operations performed in a CNN will now be given in more details. As mentioned in Section 2.1.6, CNNs comprise several arrays composed of neurons, called feature maps, each analyzing a subset from the input, which can either be 1-D or 2-D. The total number of these arrays will be noted I, indexed by i, and the number of hidden units $h_{i,j}$ in each array will be noted J with corresponding index j. As with NNs, a linear combination of input $\mathbf{x}$ with weights $\mathbf{w}_i$ is computed to obtain the values of hidden units $h_{i,j}$, except now all hidden units within an array i use the same weight vector $\mathbf{w}_i$, and the linear combination is only applied to a block of

inputs of size F, as per the following equation:

$$h_{i,j} = \sigma(\sum_{m=1}^{F} x_{j+m-1} w_{i,m} + w_{0,j}), \tag{3.34}$$

Figure 3.5 illustrates this process for a given feature map i. Since all hidden units within a given array i are computed using the same weight matrix, the feature map is the result of a filtering operation on input $\mathbf{x}_n$ with the weights $\mathbf{w}_i$, as illustrated in Figure 3.5. During training, each filter learns to detect a certain pattern in the input data. Moreover, because it scans the entire input, the location of the patterns within the input array does not matter.

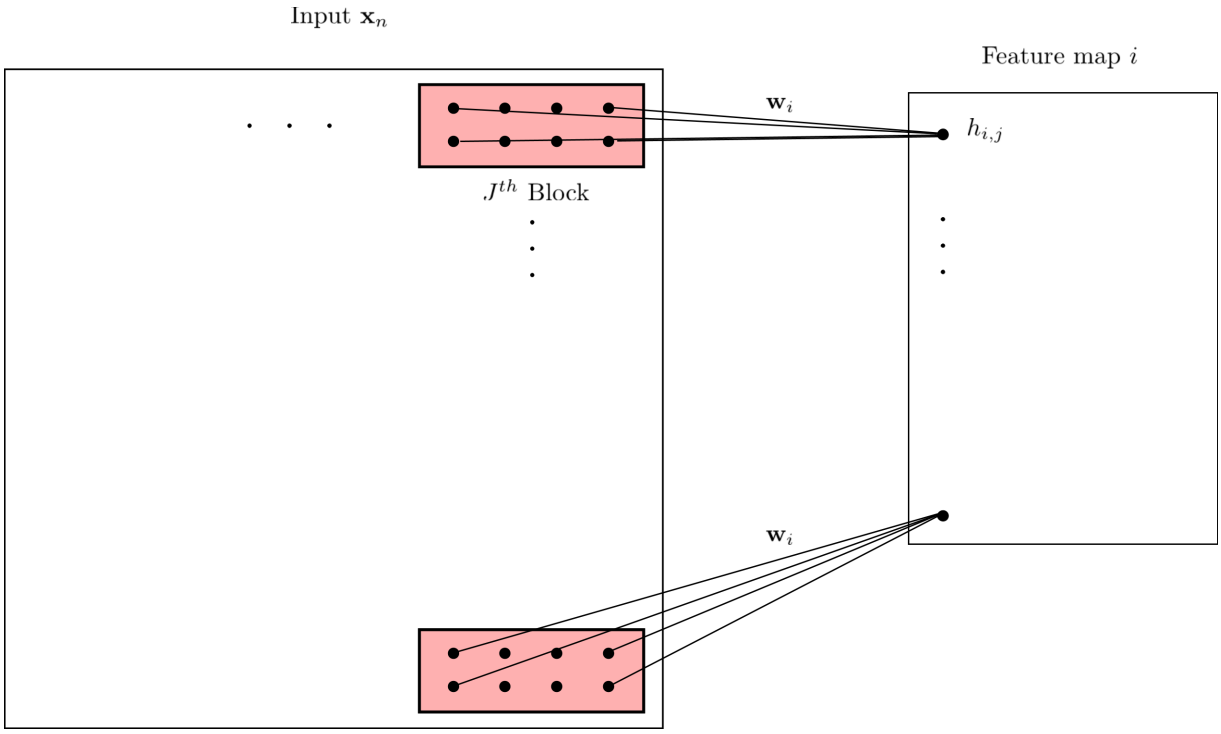

Figure 3.5: Processing of input matrix $\mathbf{x}_n$ in a CNN, to compute hidden units $h_{i,j}$ for feature map i.

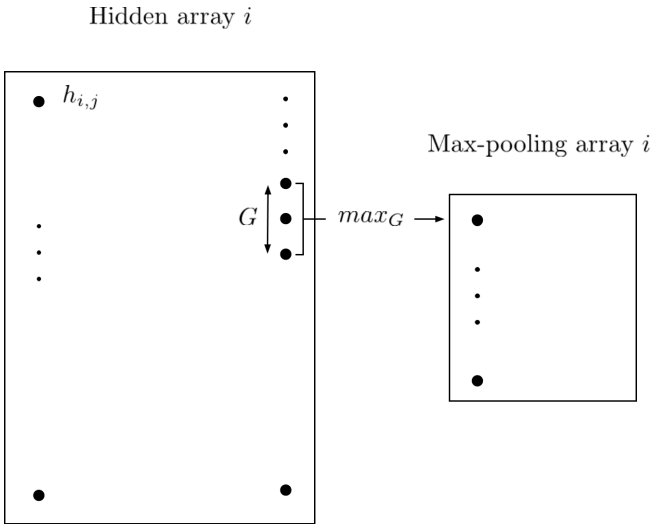

Figure 3.6: Max-pooling operation in a CNN, performed on feature map i.

After this filtering operation, the dimension of the resulting output is $I \times J$, hence it should better be reduced before being passed on to regular NN layers. This is done via a max-pooling operation, performed on each feature map, to retain only the most prominent element within a given neighborhood of M_p elements, which additionally reduces the dimensionality of the output. The operation, defined as follow, is illustrated in Figure 3.6:

$$p_{i,j} = \max_{g=1}^{G} h_{i,(j-1)G+g}, \tag{3.35}$$

where G is the pooling size.

The combination of I feature maps and I max-pooling arrays constitutes a so-called 'convolutional ply'. A deep CNN would use several such plies, on top of which one or more NN layers are added, to compute the final output, as shown in Figure 3.7.

Learning. Similarly to NNs, weight optimization in a CNN is performed via minimization of the likelihood of the target class given the input. This is done with iterative gradient descent algorithm over the weights. Although the internal structure of (D)NNs and CNNs are different, it is possible to express the filtering (weighting) operations involved in a CNN as a simple linear combination as with NNs, by combining all CNN weight vectors into a matrix in a particular way. Therefore, the gradient descent methodology of NNs is more or less straightforwardly applicable to CNNs. Details on the derivation of the CNN weights optimization can be found in [1].

3.3 Recurrent Neural Networks

As mentioned in Section 2.1.7, RNNs are NNs designed to process temporal sequences. They process an individual input vector one at a time, just like conventional NNs, but they build a time model by keeping memory of the successive inputs using a recursive scheme. This recursive scheme is implemented through recurrent connexions which enable the neurons to access the previous outputs. This was schematically illustrated in Figure 2.12, which displayed a unit with a

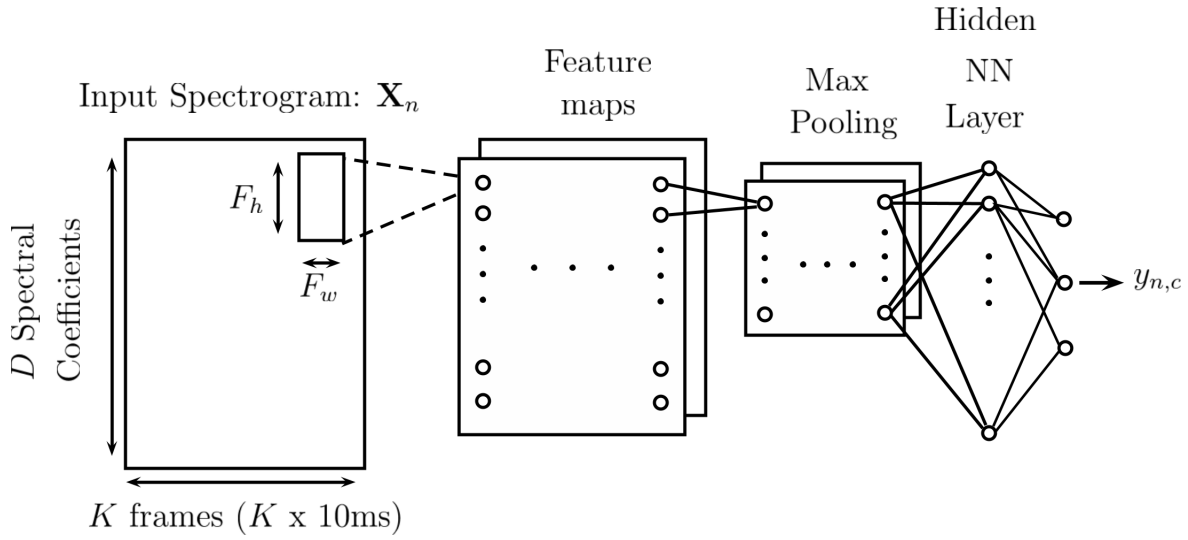

Figure 3.7: General Graph of a CNN. Here, the input is an audio spectrogram, i.e. an image composed of the concatenation of K successive spectral coefficients vectors, which illustrates how CNNs can be used to process audio data.

first-order recursion. This figure was a bit oversimplistic though, since in an actual 'conventional' RNN, the first-order recursion is implemented using the vector of outputs at the preceding layer, and not only the individual output of the unit. Indeed, formally, the equation of the j-th neuron of the hidden layer is (n denotes the time index):

$$h_{jn} = f(z_{jn}), \tag{3.36}$$

where f is a usual (non-linear) activation function, and z_{jn} comprises two terms: a linear combination of inputs $\mathbf{x}_n$ and weights $\mathbf{w}_j$, just as in regular NNs, plus a recurrent term where the outputs of the hidden layer at time $n-1$ are combined with different weights $\mathbf{r}_j$:

$$z_{jn} = \mathbf{w}_j^T \mathbf{x}_n + \mathbf{r}_j^T \mathbf{h}_{n-1} + w_{j0}. \tag{3.37}$$

Here we only consider RNNs with a single hidden layer for simpliciy, which is why no layer index l appears in the equation. The equation at the c-th unit of the output layer, at time n, is:

$$y_{cn} = g(\mathbf{v}_c^T \mathbf{h}_n + v_{c,0}), \tag{3.38}$$

where $\mathbf{v_c}$ is the vector of output weights for class c. As for (D)NNs, g is usually the sigmoid function in 2-class problems, and the Softmax function in C-class problems, $C > 2$. To sum up the different elements specific to RNNs;

- $\mathbf{y}_n$ is the output vector at time n,
- $\mathbf{h}_n$ is the hidden layer output at time n, and $\mathbf{h}_{n-1}$ is considered in the calculation of $\mathbf{y}_n$,
- $\mathbf{W}$ is the weight matrix between the input layer and the hidden layer, gathering weight vectors $\mathbf{w}_j$, $j = 1$ to J,

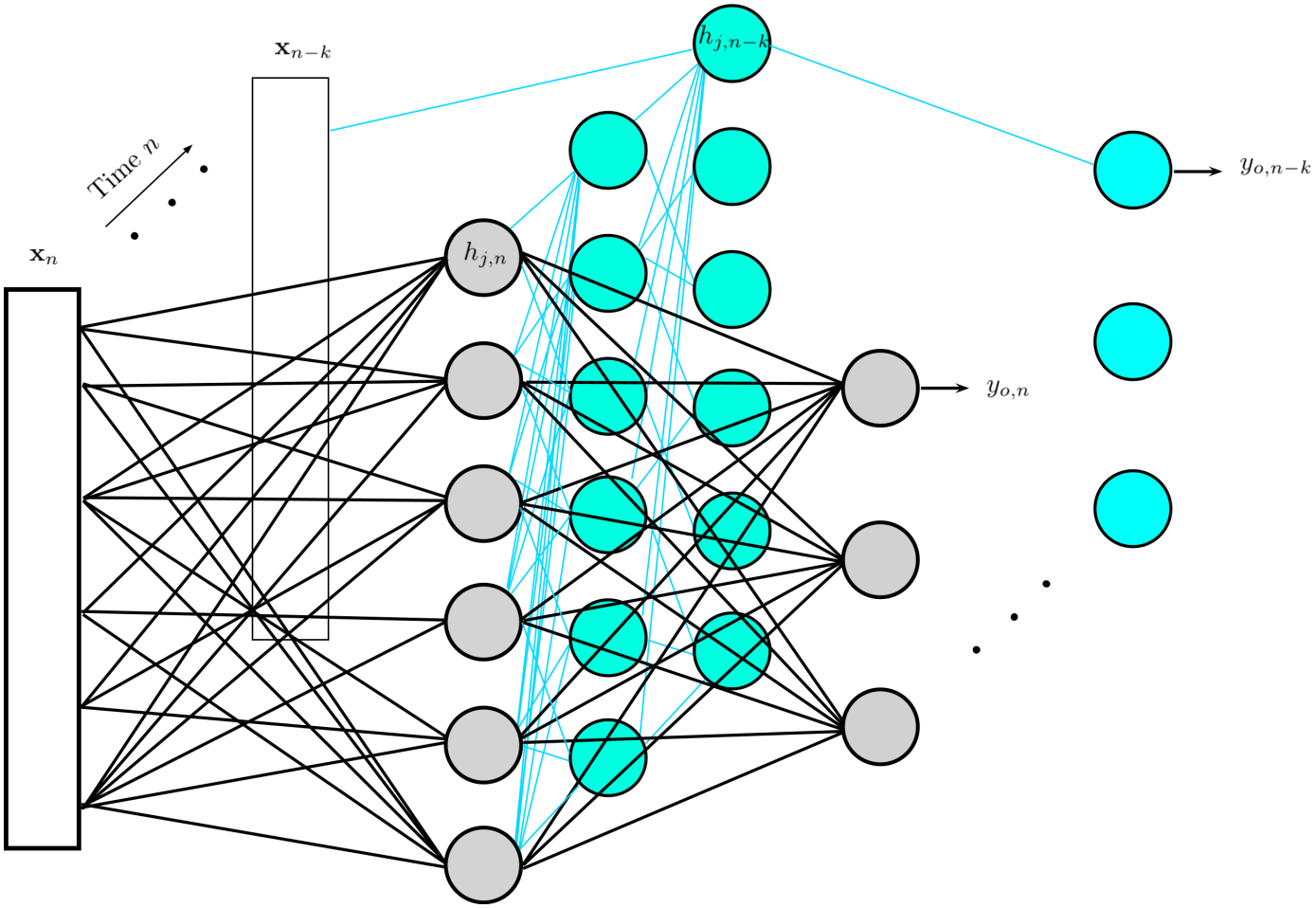

Figure 3.8: Structure of a RNN.

- $\mathbf{R}$ is the recurring weight matrix, gathering recurring weight vectors $\mathbf{r}_j$, $j = 1$ to J, and used to reinject the hidden layer at time $n-1$ into the calculation of the hidden layer at time n,
- $\mathbf{V}$ is the weight matrix between the hidden layer and the output layer, gathering weight vectors $\mathbf{v}_c$, $c = 1$ to C.

The architecture of a complete RNN is illustrated in Figure 3.8, where all past-related elements are shown in blue.

As with regular RNNs the cost function for training a RNN is the negative log-likelihood of the target variable:

$$E(\mathbf{W}) = -\log(p(\mathbf{t} \mid \mathbf{x}, \mathbf{W})), \tag{3.39}$$

evaluated on a given training dataset. Again the training process relies on gradient descent, applied on the gradient of the cost function w.r.t the weights of the network. A technique known as Back-Propagation Through Time (BPTT) [120] allows to derive a back-propagation algorithm similar to the one used for NNs, to perform gradient descent. It relies on an unfolded representation

of the RNN structure where the connection between all parameters through time can be identified. The expression for the error at the output units δ_o turns out to be very simple:

$$\delta_l = -\frac{\delta E}{\delta y_o}\frac{\delta y_o}{\delta z_o} = d_o - y_o$$

And the update rule for weights v_{jo} becomes:

$$\Delta v_{jo} = \eta \sum_n \delta_{on} h_{jn} = \eta \sum_n (d_o - y_o) h_{jn}$$

Once this first δ_o has been calculated, it can be propagated backwards into the network to update the weights $\mathbf{w}_j$ and $\mathbf{r}_j$. More details on the derivation on the BPTT formulas can be found in [120]. RNNs also suffer from the gradient vanishing or exploding problem, as mentioned in Section 2.1.7, because they are deep in time as well as in space (number of layers in the network), which adds even more complexity to the gradient propagation [54].

Long Short Term Memory

As introduced in Section 2.1.7, the LSTM network is a specific type of RNN, with units (called here cells) which are more complex than the RNN units described in the previous subsection. Indeed, an LSTM cell contains a separate state called memory state. Three different operations are performed on that state at each time-step: erase pieces of information, add some new elements, and output what is relevant. Instead of working on the input directly, the LSTM creates an intermediary vector $\mathbf{b}_n$ by combining the current input $\mathbf{x}_n$ with the previous output $\mathbf{h}_{n-1}$ into one bigger vector. The way a LSTM cell processes this bigger input can be split into three distinct stages (or operations), each of which is commonly referred to as a 'gate':

- the forget gate,
- the input gate,
- the output gate.

An illustration of the signal flow throught the different gates is given in Figure 3.9

Forget gate. In the first stage, it passes it through a sigmoid operator to decide on what, inside the current state $\mathbf{s}_n$, should be modified (what sort of information should remain in the memory and what should be forgotten). The result will then be multiplied with the cell's state in a subsequent step. This stage is run by the forget gate.

$$\mathbf{f}_n = \sigma(\mathbf{w}_f \mathbf{b}_n + w_{f0}) \tag{3.40}$$

Input gate. Separately, new information to be remembered is selected from $\mathbf{b}_n$ in three step: a first one where it is filtered through a sigmoid operator to decide what should be added into the memory, a second stage running in parallel where a copy of it is separately normalized by a hyperbolic tangent function, and a last one where the results of the previous two steps are combined

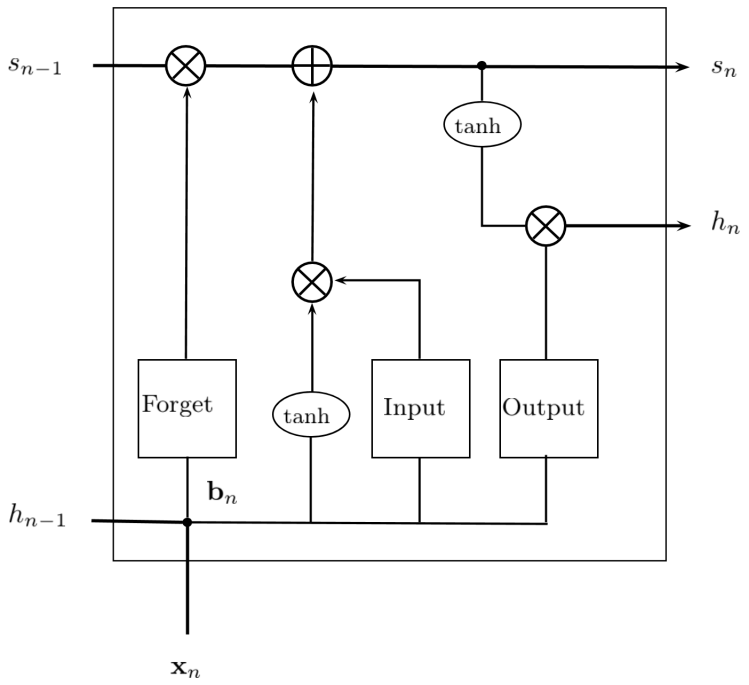

Figure 3.9: Structure of a LSTM cell.

together with the current and previous cell's state $\mathbf{s}_n$ and $\mathbf{s}_{n-1}$ to create the new cell's state. This stage is operated by the input gate (or adding gate)

$$\mathbf{d}_n = \sigma(\mathbf{W_i}\mathbf{b}_n + w_{i0}) \tag{3.41}$$
$$\mathbf{u}_n = \tanh\left(\mathbf{W_c}\mathbf{b}_n + w_{c0}\right) \tag{3.42}$$
$$\mathbf{s}_n = \mathbf{f}_n\mathbf{s}_{n-1} + \mathbf{d}_n\mathbf{u}_n \tag{3.43}$$

Output gate Lastly, $\mathbf{b}_n$ is filtered through a sigmoid function to decide what part of the cell's newly updated state ($\mathbf{s}_n$) should be output. The result of the latter operation is then multiplied by the cell's state in the output gate:

$$\mathbf{o}_n = \sigma(\mathbf{W}_o\mathbf{b}_n + w_{o0}) \tag{3.44}$$
$$\mathbf{h}_n = \mathbf{o}_n \tanh(\mathbf{s}_n) \tag{3.45}$$

In most influential papers about LSTM, the internal states of the memory cells s_n and the values of the hidden cells h_n are reset after each training sequences [40, 44, 53], stopping the accumulation of memory, following the general principle mentioned in Section 2.1.7. This allows the learning to focus on the capture of the internal temporal structure of each sequence, while preventing the network to learn any pattern in the succession of two different sequences corresponding to two different events. Similarly, a learning technique commonly used consists in truncating the propagation of gradients in time. It was shown not to affect the LSTM performance too badly [40, 53], as time-dependencies are still encoded in the memory pipe, and saves computation time. In contrast, regular RNNs lose time modeling power by truncation because propagating the error back in time

is precisely how they learn time dependencies. Therefore they can only learn dependencies that are as long as the length of the truncation.

By combining a RNN with a time decoding scheme, Graves et al. [45] solved another issue of RNNs (and NNs in general) which is that alignment between data and labels is required for training. Indeed, since they operate frame-wise predictions, RNNs need a correspond label to each input frame. In their time decoding proposal, Graves et al. [45] circumvented this by providing an estimation of the alignment within a probabilistic framework, much like HMMs [99].

3.4 Conclusion

In this chapter we gave the mathematical foundations underlying Neural Networks, starting with the basic Linear Models, which paved the way for the theoretical background behind ANNs, and the Logistic Regression unit, which is the building block of ANNs. Both are statistical models used for classification, by training their parameters to correctly output the labels of some input data from a training dataset. NNs and Deep Neural Networks can be viewed as an extension of these two models, providing more modelling power by combining multiple Logistic Regression units, resulting in a complex adaptive Linear Model. They learn some representation of the data by optimizing their parameters according to the maximization of a likelihood function, which increases their relevance to the input data. Such optimization can face the issue of local maximas of this likelihood function (or minimas of its negative) , which is why Generative pre-training was introduced to provide a better initialization of the parameters before training the network. It is based on a type of graphical models called Restricted Boltzmann Machine, and attempts to learn interactions between 2 sets of random variables.

However well trained, NNs are limited to one-dimensional inputs, which does not allow to properly handle time sequences. Another model was presented to consider 2-dimensional inputs, called Convolutional Neural Networks, where filters are used to detect pattern across both dimensions of the input signal. This is of interest to us since our target patterns are audio sequences and therefore are described by a 2-D spetro-temporal representation. However, we emphasized the limitations of CNN to fixed-size inputs, and an answer to those limitations was provided in the form of a different ANN architecture called RNN. This type of networks implements a memory to store temporal information, in order to make decisions based on it. By learning the weights associated to this memory RNNs learn the temporal structure of the training data, which will be useful for our classification task since audio sequences are characterized by the temporal evolution.

Chapter 4

Data

In this chapter we focus on data. As presented in the introduction, in the present work, the specific task we focus on is the automatic detection of shouts and screams emitted by people involved in violent events in the subway. Such events could be a person in physical difficulty, a quarrel between two or more people, panic situations, calls for help, etc. Since we opt for a supervised approach to this problem, using discriminative models (Artificial Neural Networks), we need data for both training and evaluation of these models. This kind of audio data is not common at all. In this chapter, we describe the type of data that we used in our work, the way these data were obtained, and we provide a short analysis of these data to shed a light on the complexity and 'adversity' of the transportation environment, in the present case the subway.

4.1 Data recording

To undertake the above-mentioned automatic scream/shout detection task, we need to build a database from a real 'in situation' context. Indeed detecting such a pattern as screams in a quiet acoustic environment may seem trivial with advanced machine learning techniques, but, as we will see in more details in the following, the embedded transportation context affects the signals dramatically due to its very specific sonic characteristics. In particular, as we will detail below, the alert signals to be detected are generally drowned in a quite large amount of 'ambient/background' noise and polluted by many other types of noise or interfering sources. In this thesis, we use audio data recorded during the DéGIV research project [128], which objective was the realization of a smart sensor for embedded audio and video surveillance. This project included the realization of an audiovisual database built from recordings of enacted scenes of violence in an real embedded transportation environment. From the audio recordings made during the DéGIV project we constructed our own database, as detailed in the following.

The transportation context of the audio recordings was the Paris subway, called Metropolitain, or Metro in short. More specifically, a subway train from the automatic Line 14 of the Paris Metro was reserved for the recording sessions. For data recording, the wagon was equipped with two microphones placed on the ceiling about 10 cm from one another along with a video camera. The audio recordings were made using the equipment provided by the DéGIV project. The video sequences recorded by the camera were not used in the present study, which focusses on audio signals only.

Several sessions took place between 10 am and 4 pm while the train was running its usual course, among other trains from the same subway line, running between different stations and stopping at all of them. For security matters, especially given the experimental equipment deployed in the vehicle, and most of all with respect to the simulated violence, the train did not allow regular passengers in, but only actors and extras hired for the project. For the same reasons the doors never opened at the stations to make sure that no regular passengers would get on the train, and no violent scenes were played when the vehicle was stopped at a station.

Several types of violent events were enacted. Phone snatching scenes entailed either screaming/shouting, physical/verbal altercation, or no audible clues. Quarrel scenes consisted in simple verbal altercations with verbal escalation sometimes leading to shouting, without physical violence. Lastly, fighting scenes implied some violent altercation between passengers with escalation leading to physical violence. Each scenario was played by three different sets of actors, composed of women and men alternatively victims and perpetrators. Two actors per set were used for the first scenario (phone snatching), and quarrels and fights involved two or three actors. In order

to recreate a situation as realistic as possible, numerous extras were present to simulate regular passengers. The latter were asked to react as they naturally would if facing the kind of situation described by the scene, intervening if deemed necessary and natural, thus participating in adding verbal/oral content to the scene.

Two crowd densities configurations were considered: a first one displaying a low density of passengers (between 5 and 7 people) and a second one displaying a higher density (between 12 and 17 people). Most scenes took place on a perimeter located between two adjacent gates. Two different zones were defined for the scenes to take place according to their distance to the microphone: close-distance (1 m to 1.5 m) and far-distance (3 m to 4 m). Every alert scenario was repeated for all possible combinations of set of actors, densities and zone.

Violent scenes were played and captured while the train was either accelerating, moving at stationary high speed, or braking, but not while the train was stationed as explained above. To compensate, violent scenes and sequences of doors opening and closing were also captured while the train was in the workshop. Additionally a large amount of sequences of chatter among passengers was captured. Finally, our database contains all the noises induced by the train activity i.e. doors opening and closing, brake compressors draining air, doors closing signal, etc. We believe these data convey a realistic diversity of signal occurrences for the considered application (different scenarios, different source-to-microphone positions, different noises, etc.).

4.2 Database labeling

From the available audio data we created our database by defining and broadening our own annotation. That is, the sound recordings were manually 'segmented' and cross-labelled by two different audio 'experts', namely the author of this manuscript and one of his PhD advisers. 'Segmented' means here that the boundaries of the different successive sounds were marked manually in each sound file. The first expert labelled the entire dataset and the second expert ran a second pass to validate or correct the labelling of the first expert. The total amount of labelled data was approximately 3,800 seconds.

Sounds were labelled according both to the 'speech' or 'voice' content, i.e. screams, shouts and (conversational) speech, as specified below, and to the 'environmental noise' or 'surrounding noise' content, mostly the sounds produced by the (movement of) the train. Because we led a series of different experiments with different ways of labeling the classes and exploiting the labels, we do not describe the environmental noise labels in detail here. The exact definition and content of the different classes will be provided in the next chapters which describe the different experiments conducted with different settings. We will only mention here our definition of screams, shouts and speech, which will be helpful for the insight in the signals provided in the next subsection.

Shouts are defined as loud vocal sounds with structured lexical content (aka shouted speech), whereas screams are defined as loud vocal sounds with poor lexical content, often limited to onomatopeia describing strong emotions as fear or anger for example. As we will see later in more details, scream occurrences are outnumbered by shout occurrences in our database. Because of that, in the following we often merge the scream and shout sounds into a single class called 'shout'. This class thus contains the overall set of abnormally loud sounds produced by persons subject to or witnesses of violent events. Note that in that case the term shout is used to refer to both short events, e.g., onomatopeia with high pitch and energy, and long events with speech-like lexical content, varying pitch and energy (see the examples provided in the next subsection). Finally

the 'speech' class refers to speech signals produced in 'normal' situations, i.e. mostly conversation between passengers.

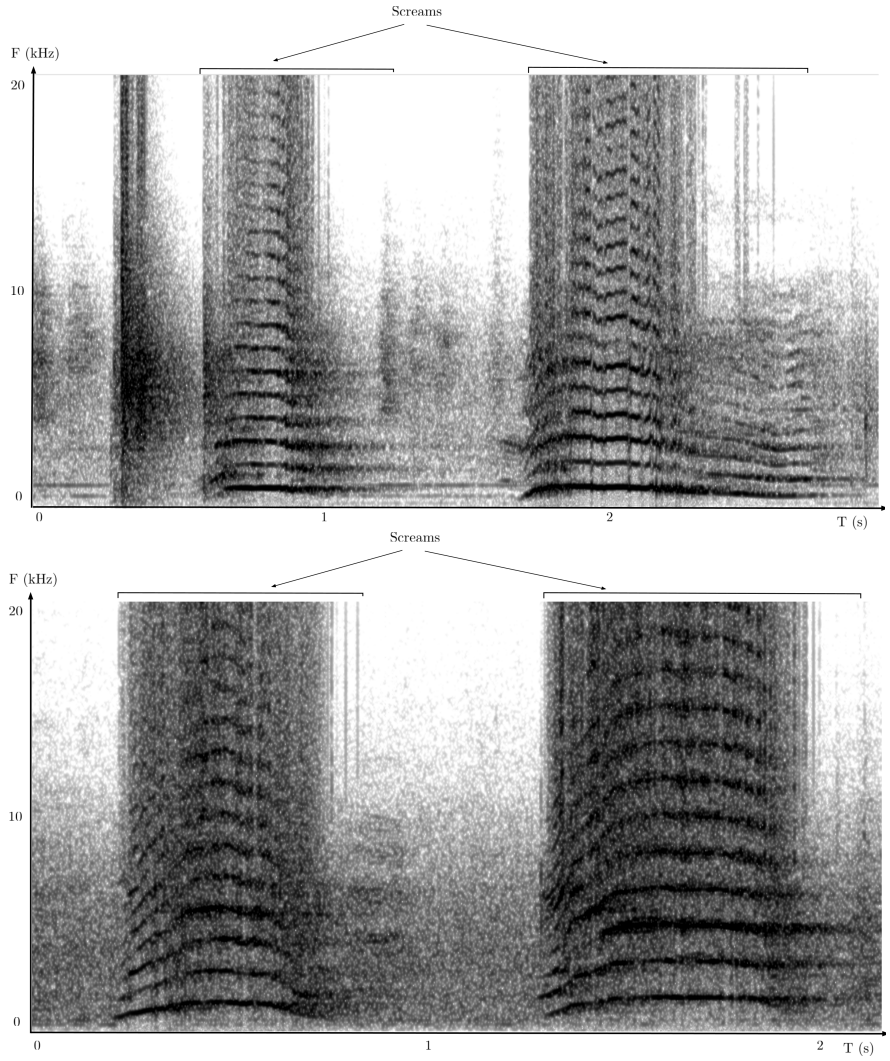

Figure 4.1: Examples of scream spectrograms with quite moderate level of background noise (two examples of two successive screams; the two examples are produced by two different persons.

4.3 A short data analysis

4.3.1 Examples of scream/shout realisations

To first illustrate the difference between screams and shouts, we provide a few example spectrograms taken from different situations where shouting and screaming occurred. Figure 4.1 shows examples of spectrograms of pure screams, i.e. screams occurring when the background noise level was quite moderate (two examples of two successive screams; the two examples are produced by

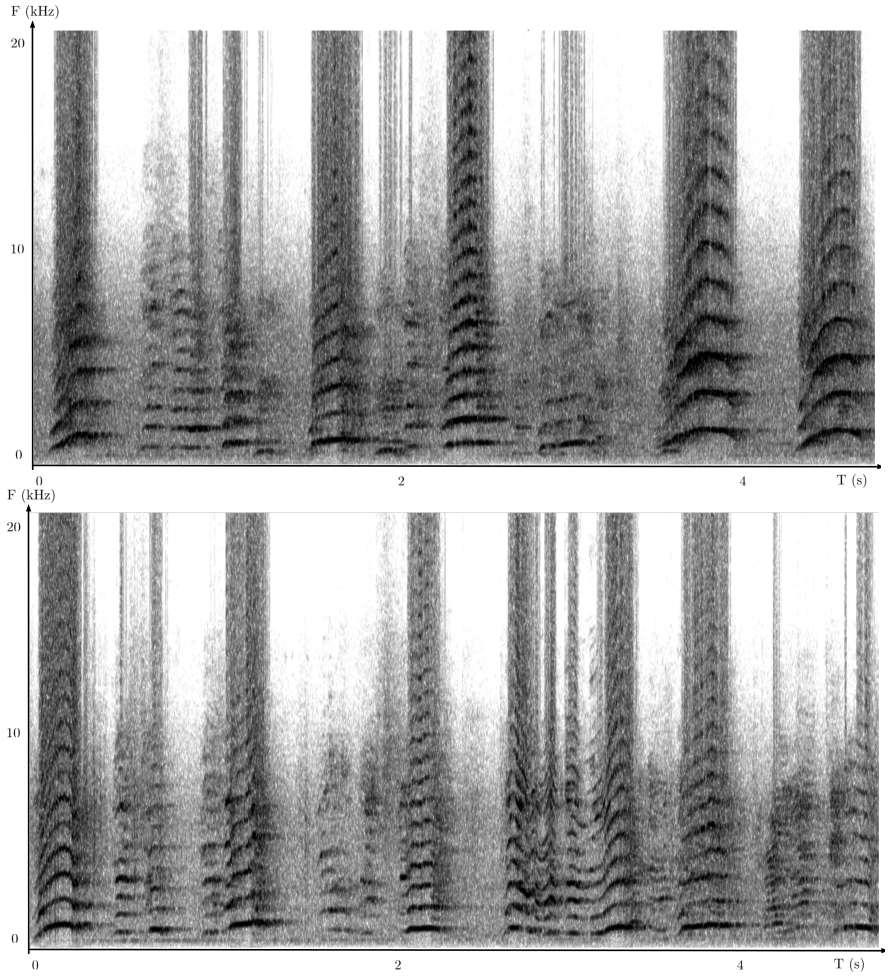

Figure 4.2: Examples of spectrograms of a series of screams produced in the context of an altercation.

two different persons). Rather neat and long harmonics can be identified, with high energy on a large range of the overall frequency band. Figure 4.2 represents the spectrogram of a series of more numerous screams produced in the context of an altercation (here also, two 'independent' examples). The patterns of the fundamental frequency and harmonics are more complex than in the previous figure, possibly indicating the presence of onomatopieas mixed with shouted speech sounds.

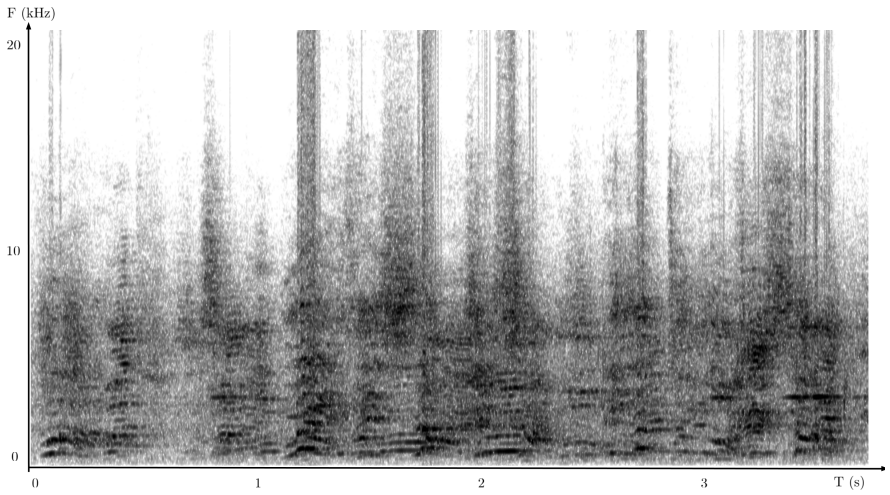

Figure 4.3: An example of spectrogram of sounds produced during a quarrel between two people.

Figure 4.3 represents about 15 s of quarrel between two people: the spectrogram becomes more intricate, with more speech-like interleaved patterns (note the reduced bandwidth occupation compared to 'pure' screams). Finally, Figure 4.4 displays two examples of spectrograms taken from two different situations where people were fighting. Here also we can see a quite complex audio scene, with speech-like, possibly interleaving, sounds, and slots occupied by scream patterns similar to the ones in Figure 4.1 (with neat and large-band harmonics).

These figures illustrate that in most cases, the violent event situation triggers various reactions from people, causing screams and shouts to be mixed to different degrees. Although such alert signals are quite specific, there exists a large variability between "speakers", and also between different realizations of screams and shouts, depending on the causing event, the emotional state, the number of persons involved, etc. This variability is one key factor that makes the automatic scream/shout detection task difficult.

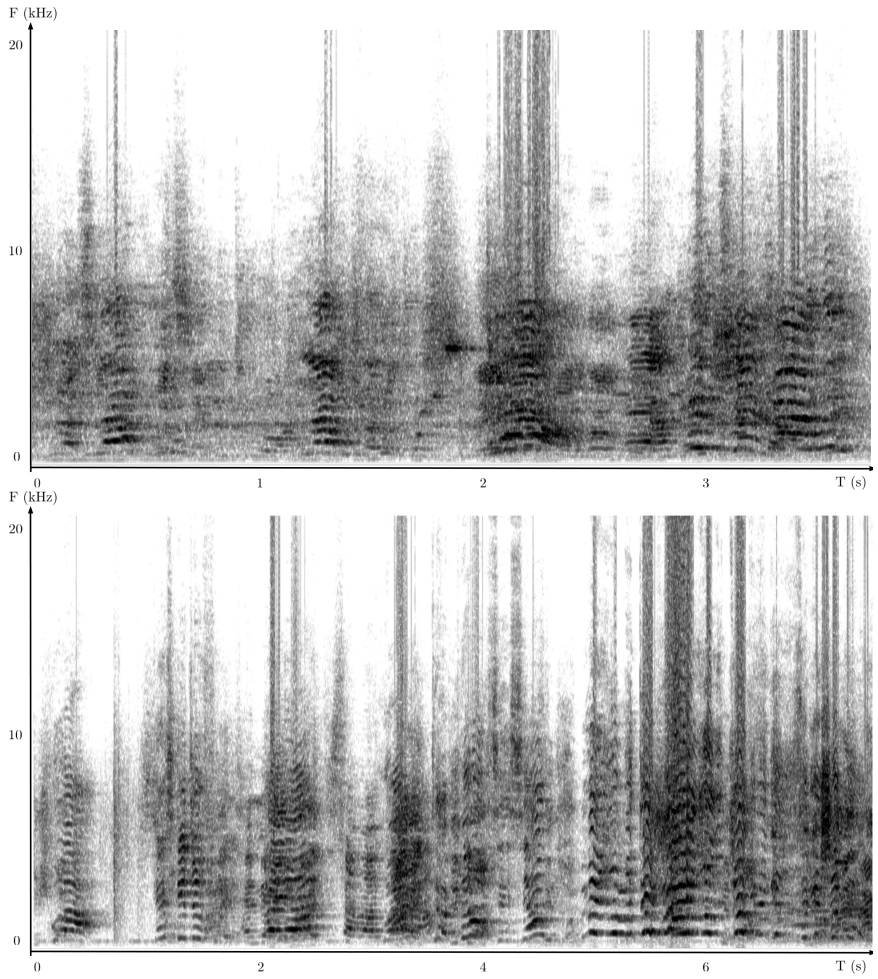

Figure 4.4: Examples of spectrograms of sounds produced during a fight between two people (including sounds produced by people surrounding the fight).

4.3.2 Acoustic properties of the subway environment

Another key factor for the present study is the fact that the embedded transport environment is in general very noisy, "rich", and strongly variable. The present study deals with a real-case environment, thus the acoustic scene contains noise from the vehicle itself (e.g., motor noise, boogie-rails frictions), noise coming from the surrounding environment (e.g., railway traffic, station noise, loud-speaker announcements), and noise produced by the passengers (in addition to speech, e.g. newspapers crumpling). In the following, depending on the sentence context, we often do not distinguish between the different noises and corresponding sources, we use the generic term 'noise' or 'background noise' to designate the set of all sounds produced by noise sources interfering with the target scream/shout signals.

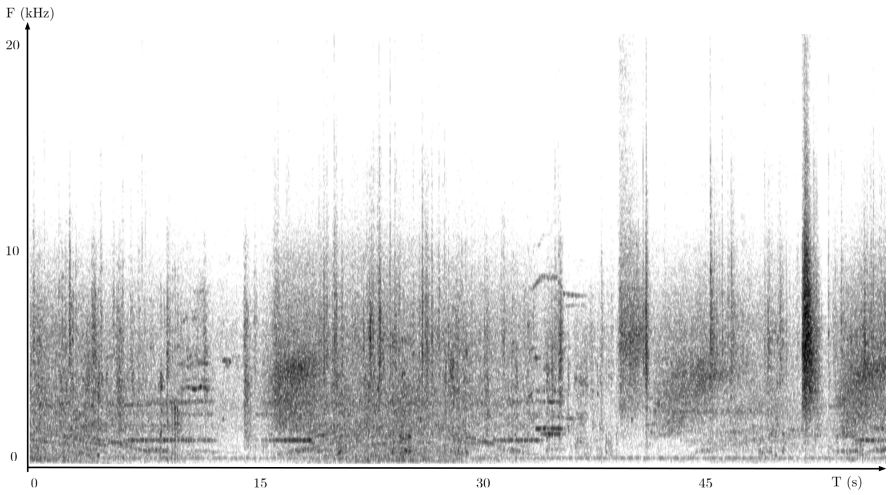

Figure 4.5: Spectral environment

Figure 4.5 shows an example of spectrogram corresponding to a 1min in-train recording spanning two arrivals at different stations and two departures. For this sound excerpt, there is no sound produced by passengers, in order to illustrate the sounds produced by the train and the environment only. The first ten seconds correspond to the arrival of the train at a station, with harmonic lines decreasing in frequency around 1kHz, followed by 2 main parallel harmonic lines at about 1kHz and 2kHz representing the squeaking noise made when the train stops, with a vertical stain starting around 5kHz up to 10kHz representative of the compressor sound. Right after a mostly blank section of a few seconds, another vertical stain from the compressor can be seen, before a horizontal line representing the same squeaking noise as before which occurs as the train goes into motion. As the train's speed increases a dark smear spanning a few seconds and frequencies in the range 5kHz-8kHz can be seen, which is characterisic of the moment before the train has entirely entered the tunnel. During the following 20 seconds a pattern in frequencies up to 5kHz can be observed, corresponding to the acceleration and deceleration of the train during the journey. Again, noises from the wheels and brakes squealing and from the compressor appear as the train stops its course around 35 s.

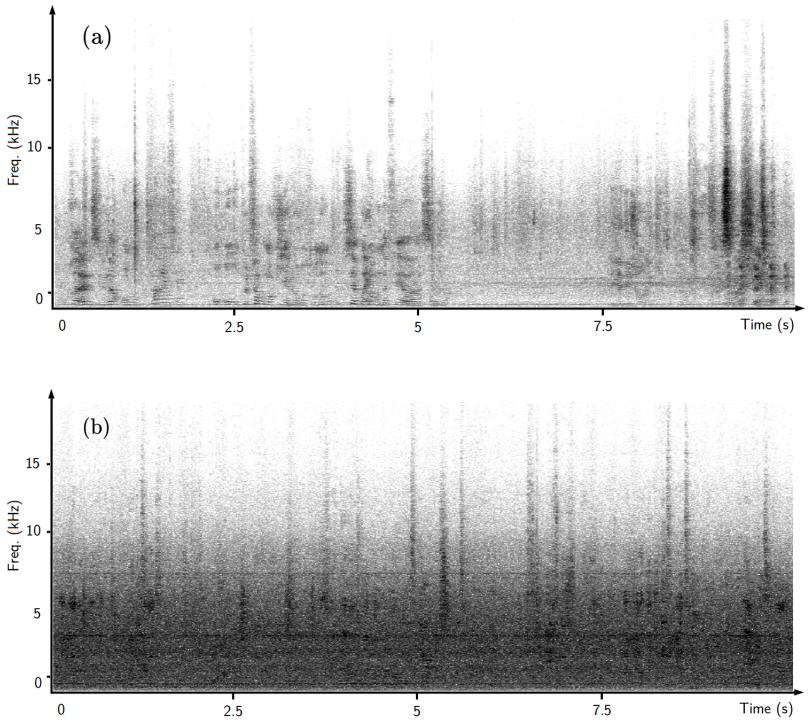

Figure 4.6: Example of spectrograms of speech when the train is in standby at a station (a) and when the train is at full speed (b).

4.3.3 Scream/shout/speech and environment noise together

Because of the continuous activity in the subway environment, a notable amount of noise is generally present in the recorded signals. This means that voice signals (scream, shout, speech) are most of the time contaminated by a significant amount of noise, possibly making the distinction between scream, shout and speech difficult, and also possibly making the distinction of these three classes with noise only very difficult, depending on the level of noise. In this subsection, we illustrate this notable characteristic of our database. Figure 4.6 displays the spectrograms of segments of speech (Figure 4.6 (a,b)) and shout (Figure 4.7 (c,d)), in a situation where the train is idle (stopped/standby at the station) (Figure 4.6 (a,c)) and in a situation where the train is at full speed (Figure 4.6 (b) and Figure 4.7 (d)). By comparing Figure 4.6 (a) (speech in idle train) and Figure 4.6 (b) (speech in train at full speed), we can see that full speed implies a much higher level of noise than idle, and consequently the speech signal is much severely masked at full-speed. Because shouts have more energy than speech, the full speed has a lower impact on shouts than on speech, as seen by comparing Figure 4.7 (c) (shout in station) and Figure 4.7 (d) (shout in full speed). Still, in Figure 4.7 (d), the shout signal is somewhat "blurred" by the full-speed noise.

To illustrate quantitatively the difference between the full speed and the idle conditions in terms of signal energy, we calculated the energy of individual 10 ms-frames extracted from typical portions of our dataset (34 s in total) and manually labelled (see Section 5.1). Figure 4.8 (a) displays the histograms of frame energy (in dB) for the two conditions (idle and full speed), for frames that

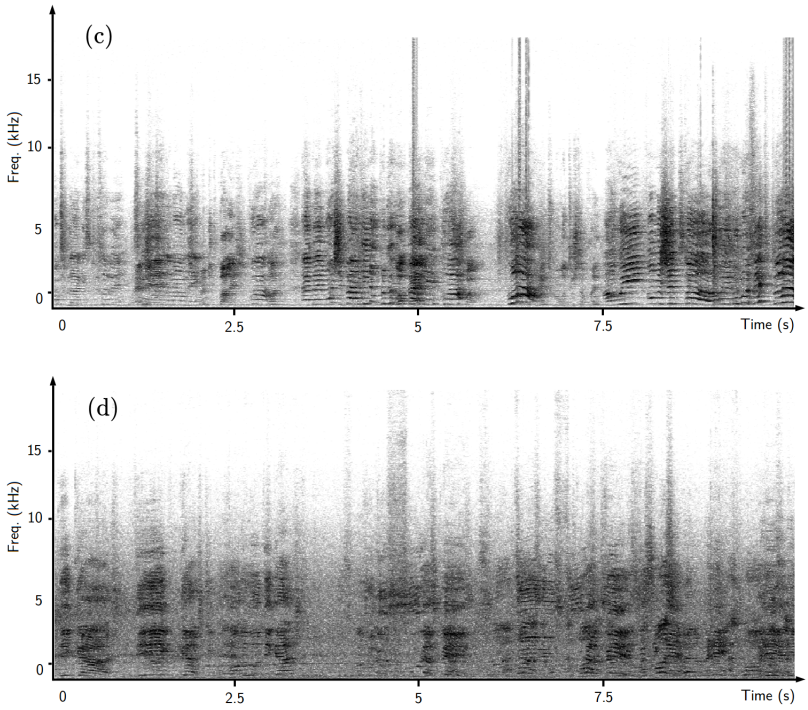

Figure 4.7: Example of spectrograms of shout when the train is in standby at a station (c) and when the train is at full speed (d).

contain no speech or shout signal (thus pure background/vehicle noise). We can see that the two histograms (therefore the two conditions) are clearly separated, with an average difference of about 60 dB, which is a large value. Figure 4.8 (b) also displays the histograms of frame energy for the two conditions, but this time for frames containing shouts. We can see that the energy of shouts clearly exceeds the energy of the background noise when the train is idle, and that the amount of additional energy from shouts spreads over a large range of values. When the train is at full speed, the (spread of) additional energy from shouts is more moderate. There is a significant overlap between the energy distributions of the full-speed noise-only frames and the full-speech noise + shouts frames. For this kind of frames, we can expect the presence of full-speed noise to make the classification into the shout category particularly difficult. For frames above a certain threshold, -20dB for this example, the classification into shouts may be easier. Note that this confirms the example spectrogram plot in Figure 4.6 (d) where some shout regions are masked by the full-speed background noise, and some other regions are more visible.

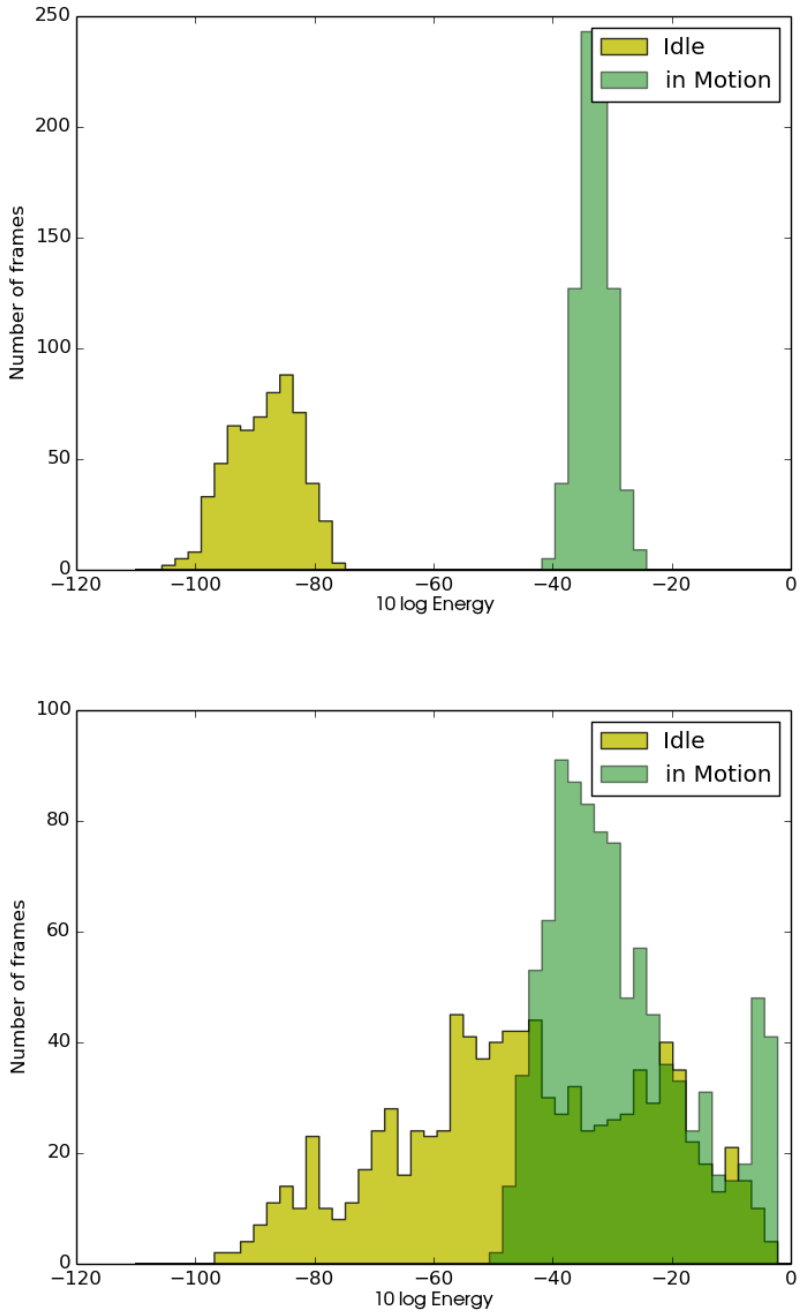

Figure 4.8: Distribution of short-term (10ms) frame energy when the train is idle vs. when it is at full speed. (a): background noise only; (b): background noise + shouts. About 34 s of manually selected portions of signal were used for this plot.

4.4 Conclusion

In this chapter we talked about the data collection process. Using Neural Networks requires a large database to be used for training (and testing). Our study being carried out in the context of audio surveillance for public transportation, we needed to build a database which was realistic enough to render the characterisics of such an environment. With the aid of the DéGiv research project we captured the audio stream from a metro in service, where some actors were re-creating situations involving violence (verbal and/or physical) between passengers, according to several scenarios. Details were given about the settings of each of the scenarios as well as the environment conditions, emphasizing the variability in audio sources. The resulting database consisted of about an hour-long audio stream captured from this realistic environment in its natural context, and was manually labelled to provide target labels to train and test the ANN models described in Chapter 3. The labels were defined relatively to the (presence of) vocal content in the signal, identifying three main patterns; screams defined as loud vocal sounds without lexical content, shouts defined as loud vocal sounds with lexical content, and speech designating conversational speech. Shouts are distinct from speech because of their higher energy, ultimately related to the degree of violence of the situation. Shouts and screams display quite different properties, but as we mentioned, occur very often together in violent situations. To illustrate this we provided spectrograms taken from the database, representing the complex mixes of screams and shouts. The presence of multiple actors taking turns in acting the different roles involved implied that the occurrences of screams and shouts in our database display a fairly high level of variability.

We also outlined the properties of our environment, as an embedded public transportation place, containing multiple sound sources with high energy and specific spetro-temporal structures. A spectrogram taken from the database was used to highlight the main events occurring in the acoustic 'background' in which our study is set. The importance of this background relatively to the voice content found in the signal from our database was illustrated by providing graphical evidence of the impact of the background on the energy level of portions of shouts and screams. As we explained in this chapter, this is one of the main difficulty linked to the realistic nature of our database, and is what makes it of particular interest in relation with the targetted application.

Chapter 5

Classification of 3 classes for the detection of shouts and screams

In this chapter, we present the first of the experimental studies that we conducted in order to automatically detect screams and shouts from an audio stream captured in the Paris Metro, using Artificial Neural Networks (ANNs). In those experiments, the problem is set as a 3-class Sound Event Detection (SED) problem. We first present the class definition, then the experimental procedure, before presenting and discussing the results.

5.1 Class Definition

To perform the Detection task we chose to use a Classification approach, where the classifier makes a prediction for every single frame from the audio stream captured by the microphones from the metro environment. Since all the models we used are discriminative, this implies that each frame in the database carry a label. Thus, our labelling strategy consisted in assigning a label to every portion of audio signal, according to the category it belonged to. Therefore the set of categories (i.e classes) that the labels were chosen from needs to fully describe the database. That is, any portion of sound from the database must fit into one of the classes defined in the set. To match this requirement, we took a hierarchical approach in the definition of our classes, identifying a list of sound events ranked by order of importance/priority. As an example, consider 2 sound events A, B ranked by order of importance, where A is considered more important than B by our approach. The first class is composed of all portions of audio containing event A, regardless of any potential concurring events, the second class is made of all portions of remaining audio signal containing event B, regardless of any potential concurring events (here no events from class A can be present since they have already been considered) and a third class is formed with all remaining audio. This way, any audio input analyzed by the classifier does belong to one class from the set of classes.

In light of the characteristics of our dataset exposed in the previous chapter, we first devised a 3-class scheme with the following target classes:

- **Shout** (screams, shouts or both; superimposed with any speech or background sounds),
- **Speech** (speech without screams not shouts; superimposed with any background sounds),
- **Background sounds** (any kind; here no speech or shouts nor screams are assumed to be present).

Our objective is here to classify the data into these three main classes. By analogy with our example on sound events A, B, screams+shouts would correspond to event A and speech to event B. The third class Background sounds therefore corresponds to the remaining audio signal and encompasses all events possibly occurring.

Shout. The first class is obviously driven by the targeted application. As already mentioned in Section 4.3, this class contains both screams and shouts, due to the low proportion of scream occurrences. This is not a major problem from the perspective of our application, since both screams and shouts are alert signals that are assumed to indicate a violent event occurring in the subway. As also mentioned in Section 4.3, scream and shout utterances can be superimposed with speech or background sounds or both. In that case, they are here considered as 'shout', since such utterances must be considered as alert signals.

Speech and Background sounds. The second class contains signals with speech, possibly contaminated by background sounds (we remind that the 'background' category encompasses all types of sounds/noise occurring in the subway environment, including the sounds produced by the train movements). This choice is motivated by the following consideration: the distinction between speech (+ background sounds) and background sounds, say, 'alone', although this term is poorly appropriate in the present case due to the variety of (possibly simultaneous) background sounds, corresponds to the most obvious distinction appearing in the data in the absence of shouts. We believe that the definition of these two classes can help the classifier in its modelling of screams and shouts (i.e the first class), as opposed to a two-category classification into **shout** (+ background sounds) vs. "**everything else**". Indeed, in the latter case, the mixing of speech with background sounds in a unique class may disturb the classifier, since the proximity between these two types of signals is questionable compared to the proximity between speech and shouts. From a broader perspective, since we adopted a hierarchical strategy for chosing our classes as explained above, our first class being quite small (the shout class) dedicating just one other class to the entire realm of all other sound events would lead to a very broad second class, which could be harder to model. Creating an 'intermediary' class is regarded here as a way to alleviate the modelling burden by providing finer discrimination between data in the learning process.

In the following, for simplicity, we refer to the shouts (+ screams) / speech / background sounds classification problem as "the 3-class problem".

5.2 Experimental settings

5.2.1 Data

We used the database described in Chapter 4, which contained a total amount of approximately 3,800 seconds of data. After labelling this database in accordance with the hierarchical strategy and the classes described in Section 5.1, this total was distributed across three different subsets: one for training, one for validation and one for testing, as shown in Table 5.1. The training set was used to train the models, the validation set was used to stop the training procedure in order to avoid overfitting problems [10], and the test set was used to provide evaluation metrics as defined in Section 2.1.9.

As can be seen in Table 5.1, our dataset suffers from imbalance, i.e. the different classes have quite different amounts of occurrences (and thus training, validation and testing data). In particular, we already mentioned that scream occurrences are quite rare. Even screams + shouts together, i.e. the shout class, is sparse compared to the speech class and the background noise class. Class imbalance is a known issue in the machine learning literature [48, 59] and it is not perfectly clear how to take this problem into account within a discriminative model framework such as the ANN framework (as opposed to generative models where it can be tackled by inserting prior class distribution in the model).

Table 5.1: Duration (in seconds) of the 3 classes in the train / validation / test datasets.

Shout	358/43/30
Speech	1554/152/356
Background	1292/190/194

However our situation is a very specific case scenario, wherein the classes are naturally imbalanced. In fact, scream/shout occurrences are even more imbalanced (under-represented) in real life than in our dataset. We have not had the chance to conduct experiments to address this problem extensively in our study but we decided to simply create a dataset with all available data, thereby displaying a higher proportion of occurrences of the shout class than would be found in fully real conditions, because the absolute number of shout/scream occurrence is already low. Indeed, the more data the more accurate the model. In short, we tried to find a trade-off between (limiting the) over-representation of screams and shouts in our dataset w.r.t. reality, and (limiting) their under-representation w.r.t. other classes. We believe that having such controlled "unrealistic" amount of data points for one class does not affect the recognition process much.

5.2.2 Metrics

The confusion matrix, defined in Section 2.1.9, will be used to present the classification results. Each input's label will be compared to the predicted class, and an error will be counted if these do not match. The percentage in the matrix will then be calculated as the number of correct matches for each class over the number of data from that class in the dataset used for testing, given by the label count.

5.2.3 Features

The data, thus labelled and distributed across three datasets (train/valid/test) as explained in Section 5.2.1, underwent a pre-processing stage where audio signal processing techniques were applied in order to extract meaningful features from the signal. The two different types of audio features used in the experiments presented hereafter are; energy-normalized log-Mel-Frequency spectral Coefficents (MFSCs) and Mel-Frequency Cepstral Coefficients (MFCCs). The former consist in the output of a filterbank applied on frames of audio signal, transposed on a special scale devised to reflect the perception of sounds by the human ears and brain, which does not match the linear frequency scale measured in Hertz (see [111] for more details). They are essentially spectral coefficients, therefore describing the frequency content of the signal, but cast on a different scale. MFCCs constitute the basis of feature extraction for audio signals. They were originally devised for speech signals, but have since been used widely in the audio signal processing community following its solid performances across a wide variety of applications (again, see [111] for more details). Both MFSC and MFCC are common representations for audio signals and their computation algorithms are provided in a wide range of feature exraction libraries, such as [12, 14, 91].

Both features are based on the spectral content of the audio signal (through $\mathbf{X}(f)$) which can contain frequencies up to a maximum value, manually set. For both MFSC and MFCC we tried out different values of this maximum frequency: 4-kHz, 8-kHz, 16-kHz, 20-kHz. The minimum frequency was set to $50Hz$. Between this minimum and maximum frequencies 40 coefficients were calculated every $T_i = 10$ ms, over a $T_w = 25$ ms time window (hence a 15 ms overlap), from the 44.1-kHz audio recordings.

5.2.4 NN settings

Having extracted features from each one of the train/valid/test subsets, we used different NN models to perform the Classification task described in Section 5.1. The first model that we used was the (direct feed-forward) NN model, described in Section 3.1.2. Since it it the simplest model

in its architecture, it will serve as a basis reference. MFCC coefficients were used as inputs for this model.

Dimensions. We tested NNs with different number of hidden layers and number of neurons in each layer, to see how the sizing of the network affects the performance: 1, 2 or 4 layers, each comprising 128, 256, 512 or 1024 units. One major general issue with NN dimensions (and how to choose them) is to avoid the so-called over-fitting problem [10]. This refers to the model's tendency to overly adjust to the training data, in such a way that it loses its generalization properties, i.e. it no longer represents new data well (data not present in the training set). This problem generally happens when the number of parameters is too large compared to the number of data, leading the model to have too many degrees of freedom when adjusting the class boundaries on the training data, making these boundaries poorly relevant to discriminate new testing data. To avoid this it is recommended to use models with a number of parameters which is a fraction of the number of available training data. This is where the Data-to-Parameters Ratio (DRP) comes into play, providing the ratio between the total number of scalar data and the total number of scalar parameters. From our experience, this ratio should preferably be equal or greater than 100. In all the following experiments, we thus limited the possible configurations of our NNs in order to avoid over-fitting as much as possible, by controlling the value of DPR. This concerns all the models we tested, not only the (D)NNs.

Context window. In order for these NNs to take into account the temporal aspect of the data, we implemented the NNs using the context window technique, as described in Section 2.1.2. The size of the context window was modified in an attempt to control the amount of temporal context provided to the classifier. The context-dependent inputs spanned a range of 10, 20 or 50 successive frames, representing respectively 100 ms, 200 ms and 500 ms of input sound.

Generative pre-training. We tried using the generative pre-training method described in Section 3.1.3, to see if it would help reach the global minimum of the cost function during the optimization process. To do this we trained a 1-layer DBN, according to the procedure described in Section 3.1.3, using 128, 256 or 512 units, generatively training it before discrimatively fine-tuning.

5.2.5 CNN

As an alternative attempt to decode temporal information from the data, we used a CNN model, described in Section 3.2. Indeed, as mentioned in the same section, CNNs are 2D models and they are capable of considering the time dimension by processing a sequence of spectral coefficient (i.e. a spectrogram) as an image. They were fed with MFSC features instead of MFCC features in order to preserve the locality property of the spectrum, as proposed in [1].

Dimensions. The rapidly growing number of parameters involved in a CNN had to be considered carefully in our experiments. Certain parameters had to be restricted to a limited range of values in order to keep the DPR high enough. More specifically, to keep the DPR above 100 we could not use more than 50 feature maps, and the max-pooling dimensions needed to be as large as 10×10 (see Section 3.2 for the definition of feature maps, max-pooling, and other CNN parameters). These two parameters had a much more significant impact on the DPR than the filters dimension or the context window size, for which we could pick values from a broad range.

Filters. To gauge the effect of the filtering process we tested different values for the 2-D filters' dimensions, varying each one separately from 5 to 20 while keeping the other one constant. This changes the size of weight vector $\mathbf{w}_i$ from Equation (3.34), which should affect the pattern detection properties of the model, besides having an impact on the DPR.

Context window. Again, different context window sizes were used to evaluate the impact of the amount of temporal context provided to the model. As for NNs, we tested 10, 20, and 50 frames, representing respectively 10 ms, 200 ms and 500 ms of input sound.

5.2.6 RNN

Ultimately, audio is made of sequences, and even if NNs with contextual windows and CNNs can be efficient in detecting patterns across time, the fact that all sequences have different durations remains a hurdle, as already mentioned in Section 2.1.2. This is why we used RNN models, presented in Section 2.1.7, to see if the performances on the target scream/shout detection improve when an explicit temporal model is considered in the NN. The RNNs used MFCC vectors as inputs and their memory covered 500-frames which is equivalent to 5s of sound.

Dimensions. We used RNNs with one single hidden layer of recurrent LSTM units, with various number of units. The RNNs we tested were using 128, 256 or 512 LSTM units.

5.2.7 Post-processing

We tested the models with and without the post-processing methods introduced in Section 2.1.8, in order to see if it improved on the 'raw' output predictions from the models. We tested the smoothing algorithm over 3 consecutive output predictions, or a majority voting over 50 ms, 100 ms or 200 ms.

5.2.8 Streaming vs pre-segmented

Some experiments were conducted with pre-segmented data, so as to be able to first measure the performance of the different models and architectures in a 'simple' offline configuration. However we wanted to test our models in the more difficult context of real-time applications, using streaming data as explained in Section 2.1.2. To this end we define 3 different configurations which we will be used throughout out different experiments:

- i) using pre-segmented data for both training and testing,
- ii) using pre-segmented data for training and streaming data for testing,
- iii) using streaming data for both training and testing.

Configuration i is unrealistic in a practical application (in the absence of automatic robust pre-segmentation which is a problem on its own), but can be used as a baseline for the other two configurations. Configurations ii and iii are realistic since online processing on unsegmented data is required only at test time. Compared to Configuration ii, tests in Configuration iii will indicate if including information about the surrounding sounds at training time is helpful to the classifier. As we justify in the following, we only tested these three Configurations with a RNN with 1 layer of 128 LSTM units and another one with 256 LSTM units.

5.2.9 Training algorithm

Cost function and weight update. To train our models as presented in Section 3.1.1, the cost function used for all our tested networks was the likelihood function, as defined in Equation (3.19), and weight optimization relied on minimization of its negative-log:

$$\begin{aligned} E(\mathbf{W}) &= -\log(p(\mathbf{t} \mid \mathbf{x}, \mathbf{W})) \\ &= -\log(\prod_{c=1}^{C} y_c(\mathbf{x}, \mathbf{w}_c)^{t_c}). \end{aligned} \tag{5.1}$$

In order to prevent weights from reaching too high values we employ Cost regularization, represented by parameters l_1 and l_2 in the following equation, which simply adds their values to the cost:

$$E(\mathbf{w}) = -\log(\prod_{c=1}^{C} y_c(\mathbf{x}, \mathbf{w}_c)^{t_c}) + l_1 \frac{1}{P} \sum_{j}^{J} \sum_{k}^{K} w_{jk} + l_2 \frac{1}{P} \sqrt{\sum_{j}^{J} \sum_{k}^{K} w_{jk}^2}. \tag{5.2}$$

where P is the total number of parameters of the model, J the total number of weight vectors $\mathbf{w}_j$ in all layers and K the number of elements in a given vector $\mathbf{w}_j$. Since a NN adjusts its weights so as to reduce the cost, it will do so in a way that also reduces the weights' values. In turn, this has the effect of avoiding that weight values soar to high and thereby avoiding over-fitting, as explained in [10]. No regularization was used for NNs and CNNs, as no over-fitting was generally observed, whereas both norm-one and square-norm regularization with $l_1 = l_2 = 0.001$ were used for RNNs to avoid that issue.

All NNs and CNNs were trained using the momentum method of the Stochastic Gradient Descent algorithm with error backpropagation, introduced in [102] and outlined in Section 3.1.2. Therefore, each weight w_{ij} was updated according to the following equation (k denotes the iteration index):

$$\begin{aligned} U_{ij}^{(k)} &= m.U_{ij}^{(k-1)} - \alpha \Delta E(w_{ij}^{(k)}) \\ w_{ij}^{(k+1)} &= w_{ij}^{(k)} + U_{ij}^{(k)}, \end{aligned} \tag{5.3}$$

where U_{ij} is an intermediary variable. The different parameters involved in this equation have distinct purposes and are described in the following.

Batches. The entire training dataset was processed in small batches, which are small subsets of the training dataset, each presented repeatedly for a given number of times (referred to as the number of epochs). The update equation was applied after each batch has been processed, averaging the values of the derivatives across the batch. For NNs, a batch size of 1024 input context windows was applied to keep the results consistent and stable across various runs. For CNNs a batch size of 256 input context window matrix was found to be enough. For RNNs we used batches of 1000 input frames.

Momentum. Parameter m in Equation (5.3), called the momentum, controls the impact of the trajectory of the parameters along the error surface, in the current parameter update. It is usually set in the range $[0, 1]$ to control the amount of the last update that should be taken into account in the current one. A momentum of 0.5 was found to be adequate for our NNs and CNNs, while no momentum was required ($m = 0$) to correctly train our RNNs.

Learning rate, epochs and stopping criterion. The learning rate α in Equation (5.3) is used to control the impact of the current derivative on the update. Indeed since each update is computed for a certain number of inputs (the current batch), it could be mis-representative of the entire dataset (especially if the current data is an outlier) and lead the gradient in a wrong direction. For NNs and CNNs the learning rate was set to 0.1 for the first 100 epochs, then decaying by a factor of 0.9 if the error delta (the difference of error, or the decrease of the objective function, between two successive iterations) was less than 0.1, and the process was completely stopped when the error delta fell below 0.01. RNNs were trained on a minimum of 50 epochs and up to 100 epochs, with a fixed learning rate of 0.05, and two stopping criterions were used: increase in the validation error of more than 5% or in the training error of more than 3%.

Dropout. The dropout method consists in ignoring some randomly selected hidden neurons in the update process. This way, only a portion of neurons adapt to the data currently being analyzed, to avoid over-fitting. This method was shown to improve the learning process in [109]. However, it was not used in the experiments corresponding to the presented results, since preliminary tests showed that no performance improvement was obtained with dropout on our dataset.

5.3 Results

All results of the afore-mentioned experiments are given in this section. Before commenting the scores presented in the different tables, it is important to mention that, because of the large amount of time required to run an experiment over the whole training database (one hour for the simplest NN and CNN and around a week for the most complex RNN), and because we had to pre-test a large number of experiments to find suitable values for parameters, these scores represent the classification results over one single experiment and should not be considered as averages over several runs (e.g. with different weights initialization). On the other hand, in order to attest to the meaningfulness of the scores, we observed that the variance between such runs generally did not exceed a few percents.

5.3.1 Feature testing

In this first experiment, we compared the use of MFSC vectors and MFCC vectors as input data. These experiments were run using a simple feed-forward NN of 1 hidden layer of 128 units, in Configuration i (pre-segmented data only). No post-processing was applied to the network's output. We first tried to feed the network with MFSCs in order to see if a simple spectral representation would suffice. Next, MFCCs were used, in the hope that the further pre-processing step represented by the DCT transform would help the network discriminate between classes. In addition, we tested different frequency bandwidths, namely upper limits of 4, 8, 16, and 20 kHz (the lower limit is 50 Hz). Table 5.2 displays the confusion matrix for the 3-class experiment using the MFSC features, for the different tested bandwidths, and Table 5.3 does the same for the MFCC features. We can see from these tables that the results are very similar in terms of performance between the two features, indicating that the DCT transform neither helps nor hampers the model. We will favor the use of MFCCs for the rest of the study, to keep in line with state-of-the-art ESR/SED techniques (see Section 2.2). As for the frequency band, it seems that broadly speaking the classification accuracy gets higher as the frequency band increases, indicating that discriminative information is present at high frequencies, as could be expected from the observations made in Chapter 4. Based on the fact that these results are not averaged over several runs and that we want to use MFCC, a frequency of 20 kHz seems to be the best choice here, and in the following we keep full-band vectors.

Table 5.2: Confusion matrices for the 3-class experiment run with a NN with 1 hidden layer of 128 units (Configuration i, no post-processing). Input features are MFSC vectors, calculated for different frequency bands.

	50-4000 Hz			50-8000 Hz			50-16000 Hz			50-20000 Hz		
1: Shout	**50.1**	40.2	9.7	**66.1**	26.4	7.5	**71.5**	24.9	3.6	**62.4**	31.8	5.7
2: Speech	1.1	**72.9**	26.1	1.7	**65.8**	32.5	2.1	**73.7**	24.2	1.1	**72.3**	26.6
3: Background	0.5	28.1	**71.5**	0.6	16.4	**83.1**	0.5	25.2	**74.3**	0.3	19.4	**80.2**

Table 5.3: Confusion matrices for the 3-class experiment run with a NN with 1 hidden layer of 128 units (Configuration i, no post-processing). Input features are MFCC vectors, calculated for different frequency bands.

	50-4000 Hz			50-8000 Hz			50-16000 Hz			50-20000 Hz		
1: Shout	**56.4**	32.9	10.8	**66.1**	28.9	5.0	**69.4**	26.4	4.2	**71.1**	25.2	3.6
2: Speech	2.2	**72.5**	25.2	2.5	**75.4**	22.1	1.9	**75.7**	22.4	2.3	**73.1**	24.6
3: Background	1.7	37.3	**61**	1.0	31.3	**67.7**	1.1	27.5	**71.4**	1.1	23.6	**75.3**

5.3.2 NN

Varying the NN dimension. Using the Full-band MFCC vectors as input, we now try to modify the NN's architecture. Table 5.4 shows the confusion matrices obtained with the basic NN with one single hidden layer, for different number of units in this layer. Here also, these experiments were led on the pre-segmented mode, i.e. in Configuration i, and no post-processing was applied to the network's output. We can see that, a bit surprisingly, increasing the number of units in the hidden layer does not significantly impact the overall performance of the classifier. The lowest tested number of hidden units, 128 , seems to do the job as well as higher values. In fact, using 1024 hidden units does affect the result but not in a good way, as it severely degrades the detection of the speech class. This could be an illustration of the over-fitting problem. In the present case (one layer of 1024 hidden units), the DPR value is 58, which is not very large, but may still be considered acceptable, which is why it is not a great indicator of possible over-fitting. To further check on this problem, Table 5.5 provides the confusion matrix of the 1×1024 NN as calculated on the training data. We can see that the classification accuracy (diagonal values of the confusion matrix) on the training data is above 90% for all classes with a total error as low as 11.1%, while the accuracy is only 50.7% for background sound class on the test set. Such a drastic difference is strongly reflects the over-fitting problem: the NN is able to correctly discriminate the training data while poorly generalizing to test data. Per these conclusion, in the following we consider NNs with layer(s) of 128 units (note that the corresponding DPR is 467).

After trying different numbers of neurons, we tried to increase the NN's modelling power by adding hidden layers. Table 5.6 shows the effect on the performance for a fixed number of units on each layer (128), using 2 and 4 hidden layers. Here also, no notable performance gain seems to appear when using more and more complex ('deep') architectures. In fact the results are quite comparable for the various numbers of layers, and only vary by a few percentage without any noticeable trend in improvement or degradation. When a gain is observed for one class, it is generally compensated by a loss for another class. DPR values for 2 and 4 layers (respectively 354 and 239) would suggest that these results are not due to over-fitting, however a look at the confusion matrix obtained on the training data after the last epoch tells us that a loss of generalization power is actually occurring when increasing the number of hidden layers. Indeed, all three models fit to the data very similarly, despite the significant drop in accuracy on the test data for the NNs with 2 and 4 layers, as shown in Table 5.7.

Table 5.4: Confusion matrices for the 3-class experiment run with a NN with 1 hidden layer (Configuration i, no post-processing, MFCC input vectors). The number of neurons in the hidden layer is chosen from $\{128, 256, 512, 1024\}$.

	1×128			1×256			1×512			1×1024		
1: Shout	**71.1**	25.2	3.6	**68.0**	27.3	4.7	**65.9**	30.3	3.8	**71.3**	17.0	11.7
2: Speech	2.3	**73.1**	24.6	2.0	**70.4**	27.6	2.6	**78.3**	19.2	3.5	45.7	**50.7**
3: Background	1.1	23.6	**75.3**	1	20.4	**78.6**	1.8	28.3	**69.9**	1.1	3.7	**95.2**

Table 5.5: Confusion matrix for the 3-class experiment run with a NN with 1 hidden layer of 1024 neurons, obtained on the training set after the last training epoch.

	training error		
1: Shout	**95.8**	3.0	1.2
2: Speech	0.2	**90.4**	9.5
3: Background	0.0	8.7	**91.2**

Table 5.6: Confusion matrices for the 3-class experiment run with NNs with 1, 2 and 4 hidden layers of 128 units each (Configuration i, no post-processing, MFCC input vectors).

	1×128			2×128			4×128		
1: Shout	**71.1**	25.2	3.6	**68.2**	28.5	3.3	**67.7**	27.5	4.9
2: Speech	2.3	**73.1**	24.6	1.8	**77.6**	20.6	2.3	**74.5**	23.2
3: Background	1.1	23.6	**75.3**	1.16	28.5	**70.3**	1.2	25.2	**73.5**

Table 5.7: Confusion matrices obtained on the training data after the last training epoch, for the 3-class experiment run with NNs with 1, 2 and 4 hidden layers of 128 units each (Configuration i, no post-processing, MFCC input vectors).

	1×128			2×128			4×128		
1: Shout	**76.0**.7	19.7	3.6	**77.8**	18.6	3.6	**77.2**	19.2	3.6
2: Speech	2.3	**78.7**	19.0	2.2	**79.3**	18.4	2.3	**78.2**	19.4
3: Background	0.5	17.3	**82.2**	0.5	17.2	**82.2**	0.5	17.6	**81.9**

Context window. So far the input was created using context windows of 10-frames, which only provides the model with 100 ms of temporal context. Increasing the size of the context window will provide longer temporal context.

Table 5.8 shows the effect of different window lengths on a DDN with 1 layer of 128 units. In contrast with the previous two experiments, a clear effect is noticed here. As the context window grows larger, the detection accuracy seems to largely improve for the shout and speech classes, while diminishing to a lesser extent for the background class. This improvement for two classes out of three has to be put in perspective by recalling that because we are using pre-segmented data, implying that all occurrences shorter than the length of the context window are discarded from the training, validation and test datasets. As a result, the shortest occurrences in the datasets match the window length, i.e respectively 200 ms and 500 ms for window lengths of 20 and 50. The

Table 5.8: Confusion matrices for the 3-class experiment run with a NN with 1 layer of 128 units (Configuration i, no post-processing, MFCC input vectors), for different sizes of context window (10, 20 and 50 frames).

	10 frames			20 frames			50frames		
1: Shout	**71.1**	25.2	3.6	**74.2**	22.7	3.1	**82.6**	14.1	3.3
2: Speech	2.3	**73.1**	24.6	1.6	**78.7**	19.6	1.0	**83.6**	15.4
3: Background	1.1	23.6	**75.3**	0.8	26.1	**73.1**	0.6	27.9	**71.4**

Table 5.9: Total weighted accuracy from the three-class results for NN with 1 layer of 128 units (Configuration i, no post-processing, MFCC input vectors), for different sizes of context window (10, 20 and 50 frames).

	10 frames	20 frames	50 frames
Accuracy	**73.7**	**76.6**	**79.5**

average occurrence length is 1.83 s for the shout class and 2.03 s for the speech class. Considering this, getting rid of the shortest occurrences amounts to discarding the outliers from the dataset, which helps the classifier because it reduces the variance of each class. The decrease in accuracy for the background class can be explained by the fact that the temporal structure of the events in this class is less pronounced than for class speech and shout, which have strong time signature and therefore benefit more from adding temporal context information. Overall the NN using 50-frames inputs is our best choice since it provides the highets score on classes shout and speech, with over 10% increase over using 10-frames context window for both class, although the score on class background decreased by 4%. The averaged error, weighted according the class distribution in our test dataset, for the three experiments from Table 5.8 is provided in Table 5.9, which clearly shows the improvement as the size of the context window increases.

Generative pre-training. Up until now, the weights in our networks were initialized randomly. A better initialization may help find the global minimum (or a better local minimum) of the cost function. According to the theory presented in Section 3.1.3 we set out to initialize the weights of our network by a generative pre-training method. To this end, we considered one hidden layer as a RBM layer and trained it as such using contrastive divergence as introduced in [52]. The final weights were then used as initial values to fine-tune the layer using discriminative training. Table 5.10 shows the results of such training for different layer sizes. Comparing it with the results of a regular NN without pre-training (see Table 5.4) shows that, in the present experiments, pre-training does not seem to improve learning, meaning that generative pre-training does not help find a significantly bette local minimum of the cost function. The layer with 512 units yielded the best results overall.

Post-Processing (smoothing and majority voting). To improve on the consistence of classification over time (i.e. avoid possible 'erratic' output sequences), we applied the two post-processing algorithms introduced in Section 2.1.8 on the output predictions from our NNs. Table 5.11 puts in light the difference before and after smoothing the predictions of a NN with 1 layer of 128 hidden units on inputs formed with a 10-frame context window. In this experiment, the smoothing leads to a consistent improvement of the accuracy on all 3 classes. Table 5.12 puts in light the difference before and after majority voting, for different lengths of the majority voting window. Here also, majority voting improves the accuracy, and from a broader perspective the larger the segment the higher the overall improvement, although it can slightly vary across the classes. For either a 100 ms or a 200 ms window, the accuracy gain is quite impressive, significantly superior to the gain obtained with smoothing. Also, the results for the 200 ms window in Table 5.12 are very different from those in Table 5.8 with improvements in accuracy for all classes(+0.8% for 'shout', +3.1% for 'speech' and +9% for background). All these experiments suggest that in a real-world implementation of such a system it is possible to increase the final accuracy of the system by post-processing

Table 5.10: Confusion matrices for the 3-class experiment run with pre-trained NNs of 1 layer (Configuration i, MFCC input vectors), without post-processing.

Architecture:	1×128			1×256			1×512		
	0	1	2	0	1	2	0	1	2
0: Shout	**60.7**	31.8	7.48	**60.2**	34.6	5.22	**66.6**	28.9	4.52
1: Speech	2.07	**74**	23.9	1.58	**82.6**	15.8	2.16	**75.5**	22.4
2: Background	0.51	35.9	**63.6**	0.62	43.4	**56**	1.19	29.4	**69.5**

Table 5.11: Confusion matrices for the 3-class experiment run with a NN with 1 hidden layer of 128 units (Configuration i, MFCC input vectors), before and after post-processing of the output stream with the smoothing algorithm.

	no smoothing			smoothing		
1: Shout	**71.1**	25.2	3.6	**71.8**	24.5	3.6
2: Speech	2.3	**73.1**	24.6	1.3	**75.8**	22.9
3: Background	1.1	23.6	**75.3**	0.7	21.1	**78.2**

Table 5.12: Confusion matrices for the 3-class experiment run with a NN with 1 hidden layer of 128 units (Configuration i, MFCC input vectors), before and after post-processing of the output stream with the majority voting algorithm, for different segment lengths.

	no maj.vote			maj.vote: 50ms			maj.vote: 100ms			maj.vote: 200ms		
1: Shout	**71.1**	25.2	3.65	**69.8**	28.4	1.7	**78.3**	16.7	5.0	**75.0**	12.5	12.5
2: Speech	2.3	**73.1**	24.6	0.9	**77.9**	21.2	0.3	**81.0**	18.7	0.3	**81.8**	17.9
3: Background	1.1	23.6	**75.3**	0.5	20.2	**79.3**	0.3	22.1	**77.6**	0.0	17.9	**82.1**

the output stream.

The behavior of the majority voting algorithm is illustrated in Figure 5.1, Figure 5.2 and Figure 5.3, which each represent a portion of input signal (top), corresponding 'raw' network output predictions (middle), and output predictions corrected by the post-processing algorithm (bottom). In the examples displayed in Figure 5.1 and Figure 5.2, the majority voting algorithm does correct most errors and lead to better accuracy. However, as can be seen in Figure 5.3, the algorithm also has its own flaws. In particular, when the model makes more wrong predictions than correct ones, the majority voting algorithm cancels out the good predictions and keeps the bad ones. This is because it relies on the assumption that the classifier is inherently good, in the sense that it will necessarily output more correct answers than wrong answers on the considered segment. When it does not, the algorithm's impact is obviously negative.

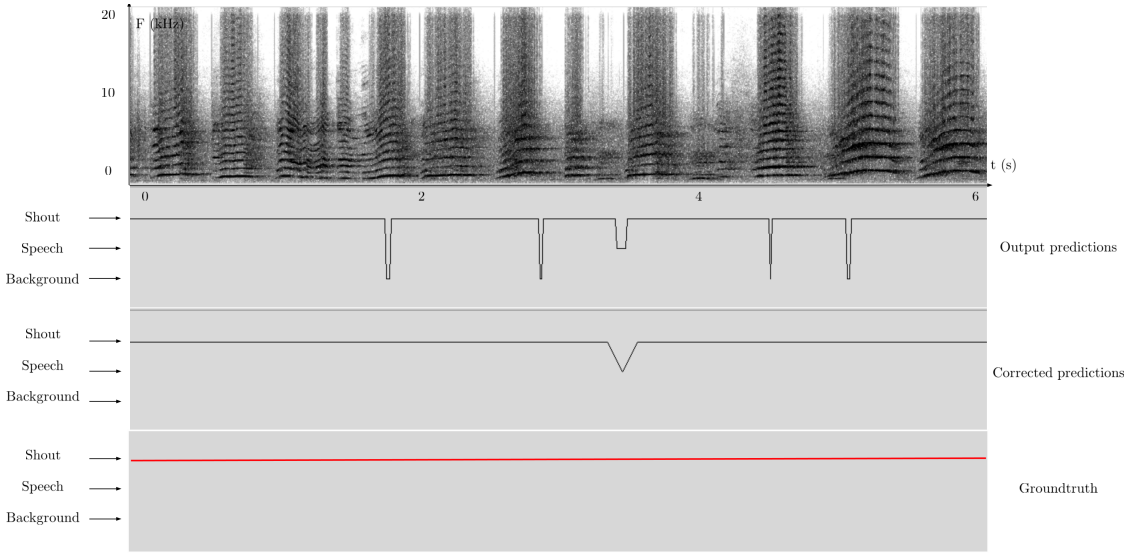

Figure 5.1: Illustration of the detection without and with the majority voting post-processing algorithm over 200 ms, on a portion of shout signal (1×128 NN, Configuration i, MFCC inputs).

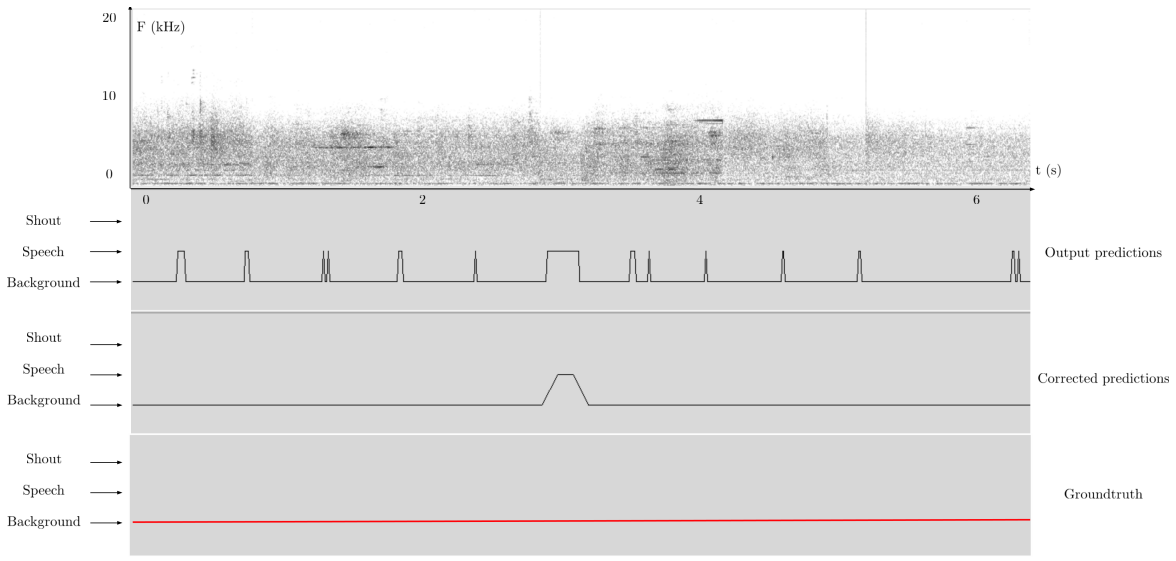

Figure 5.2: Illustration of the detection without and with majority voting post-processing algorithm over 200 ms on a portion of background noise. (1×128 NN, Configuration i, MFCC inputs)

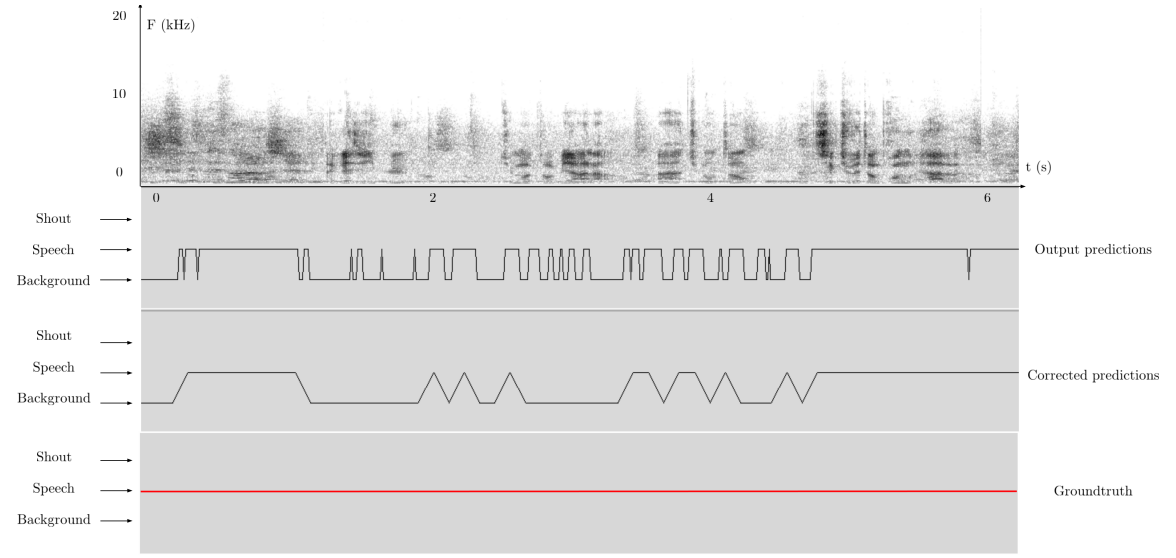

Figure 5.3: Illustration of the detection without and with the majority voting post-processing algorithm over 200 ms on a portion of speech (1×128 NN, Configuration i, MFCC inputs).

5.3.3 CNN

We now present the results that we obtained with the convolutional neural network model. We will present the results obtained in Configuration i and without any post-processing, and with a single NN output layer of 128 units. Our first experiment with the CNN consists in modifying the length of the context window. We arbitrarily decide to set the filter dimension to 5×5. The

Table 5.13: Confusion matrices for the 3-class experiment run with a CNN with 1 hidden layer of 5×5 filters,, plus 1 output NN layer of 128 units (Configuration i, no post-processing), for different context window lengths.

	10 frames			20 frames			50 frames		
DPR	137			185			137		
1: Shout	**65.3**	31.3	3.4	**71.8**	26.8	1.4	**81.6**	16.3	2.0
2: Speech	0.7	**75.0**	24.3	1.2	**75.4**	23.4	1.3	**79.4**	19.3
3: Background	0.8	23.2	**75.9**	0.7	22.8	**76.6**	1.8	32.5	**65.7**

Table 5.14: Total weighted accuracy from the three-class results for CNN with 1 hidden layer of 5×5 filters, plus 1 output NN layer of 128 units (Configuration i, no post-processing), for different context window lengths.

	10 frames	20 frames	50 frames
Accuracy	**74.8**	**75.6**	**74.9**

max-pooling dimension had to be adjusted for each context-window in order to keep the DPR value above 100. Therefore for 10-frames context windows we used max-pooling over 5×5 , for 20-frames context window we used 7×7 max-pooling, and for 50-frames we used 10×10 max-pooling, which resulted in decent DPR values ($DPR >= 100$). The number of feature maps was set 50 as suggested in [1]. Table 5.13 shows that the 50-frames context window leads to much better accuracy for classes shout and speech, as the CNN seems to be able to better grasp the temporal structure of the data. This is consistent with what was observed for the NN with the context window in Table 5.8. Table 5.14 shows that the total averaged accuracy is pretty similar for all three context window sizes, which is why we decided to keep the 50-frames context window option since it yielded better results for class shout.

Based on this result, using a 50-frames window, we tested a set of filters dimensions, modifying one dimension after the other, using a single convolutional hidden layer (with a max-pooling dimension of 10×10 as before), plus a single NN layer of 128 units. Table 5.15 shows the results obtained when varying the filter size along the frequency axis, the most satisfying results being obtained with a length of 15, with a drop in performance when going from 15 to 20. This suggests that increasing the filter size on the frequency dimension helps recognize patterns from classes speech and background sound. Table 5.16 shows that increasing the filter size along the time axis results in higher accuracy for the speech class, as before, however this time impairs that of class background. Class shout does not seem to be affected except for a filter of 15×5 where its detection accuracy is lowered by 10%.

Now, considering the two dimensions at the same time we tried varying them both. Results are reported in Table 5.17, showing lower accuracy for class background souds, when compared with Table 5.15.

By the look of Tables 5.15, 5.16 and 5.17 we conclude that the best performing setting was with a filter of 5×15. Its accuracy score are higher than the 1-layer NN with 128 hidden units from Table 5.8 for classes shout (4%) and background sound (6%) but are slightly lower for class speech (2%), with an overall better performance.

Note that to avoid overfitting as much as possible, we carefully monitored the DPR, whose values are reported in the tables. That the reported values are quite decent (above 130) and increase

with the filter size. This is because in the CNN implementation no zero-padding was used in the convolution process, which makes the output of the operation smaller than the input (by the size of the filter). Therefore for larger filters the dimension of the hidden feature map was smaller, thereby reducing the number of parameters.

Table 5.15: Confusion matrices for the 3-class experiment run with a CNN with 1 hidden layer of filters and 10×10 max-pooling, plus 1 output NN layer of 128 units (Configuration i, 50-frame context window inputs, no post-processing), for a filter size along the time dimension equal to 5 and different filter sizes along the frequency axis.

	5×5			5×10			5×15			5×20		
DPR	137			135			195			191		
1: Shout	**81.6**	16.3	2.0	**83.7**	14.3	2.0	**85.7**	12.2	2.0	**73.5**	24.5	2.0
2: Speech	1.3	**79.4**	19.3	1.4	**81.8**	16.8	1.9	**81.6**	16.5	0.6	**79.1**	20.3
3: Background	1.8	32.5	**65.7**	1.2	29.8	**69.0**	1.8	20.5	**77.7**	1.8	23.8	**74.4**

Table 5.16: Confusion matrices for the 3-class experiment run with a CNN with 1 hidden layer of filters and 10×10 max-pooling, plus 1 output NN layer of 128 units (Configuration i, 50-frame context window inputs, no post-processing), for a filter size along the frequency dimension equal to 5 and different filter sizes along the time axis.

	5×5			10×5			15×5			20×5		
DPR	137			135			175			171		
1: Shout	**81.6**	16.3	2.0	**85.7**	14.3	0.0	**75.5**	24.5	0.0	**85.7**	14.3	0.0
2: Speech	1.3	**79.4**	19.3	2.5	**84.6**	12.9	0.5	**89.3**	10.2	2.5	**87.3**	10.2
3: Background	1.8	32.5	**65.7**	2.1	42.8	**55.1**	0.0	44.0	**56.0**	1.2	47.3	**51.5**

Table 5.17: Confusion matrices for the 3-class experiment run with a CNN with 1 hidden layer of filters and 10×10 max-pooling, plus 1 output NN layer of 128 units (Configuration i, 50-frame context window inputs, no post-processing), varying the filter size across both dimensions at the same time.

	10×10			15×15			20×20		
DPR	131			216			183		
1: Shout	**79.6**	20.4	0.0	**75.5**	22.4	2.0	**79.6**	18.4	2.0
2: Speech	0.3	**87.6**	12.1	1.3	**80.2**	18.6	0.9	**85.1**	14.0
3: Background	0.6	34.6	**64.8**	1.5	38.5	**59.9**	1.2	43.4	**55.4**

5.3.4 RNN

Finally, we report the results that we obtained with a recurrent model, using LSTM units as described in Section 3.3. Here no post processing was applied to the network predictions, in order to compare the network's inherent performance with those of NNs and CNNs.

Table 5.18 shows the confusion matrices for different number of units in the hidden layer (128, 256 and 512). Interestingly, the accuracy on class shout decreases as the number of units grows (from 86.7% to 70.2%), while the exact opposite happens for class shout and background (from 73.5% to 81.1% for shout and from 82.5% to 95.4% for class background). Overall, not much gain is drawn from the more complex configurations compared to the 128-unit RNN, but the DPR indicates that the RNN with 256 and 512 units are likely to be subject to overfitting, which may explain why. Despite that, the 256-unit architecture seems to handle the task better overall, with very consistent accuracy across all three classes (between 78.1% and 83.1% as seen in Table 5.18), as compared to the NN which displayed much bigger differences across classes (accuracy between 71.4% and 83.6%), and even the best CNN from Table 5.15 (accuracy between 77.7% and 85.7% with filters of 5×15). This shows that the temporal abilities of the RNN allows it to better model all classes, despite the fact that the best CNN was able to achieve better recognition accuracy on class shout (85.7% vs 78.1%).

On another note, looking at the training results gives us a other insight on the model's ability to fit on the dataset. More specifically, Table 5.19 displays the confusion matrix obtained on training data after the last training epoch by a RNN with 1 layer of 256 LSTM units and our best performing CNN from Table 5.15 (on 50-frame context windows, with filters of 5×15), showing RNN's significantly higher capacity to learn from the dataset. It also sheds light on the amount of over-fitting that our RNN suffers from, despite the quality of the final results, which is of course related to the low DPR observed for the RNN. The difference between this confusion matrix and that obtained on the test data (as shown in Table 5.18) suggests that a better parameterization of the learning process might yield better results by reducing the effect of over-fitting.

Table 5.18: Confusion Matrix obtained on training data by a RNN with 1 layer with different number of hidden units (Configuration i, no post-processing).

	128			256			512		
DPR	152			34			9		
1: Shout	**73.5**	21.4	5.2	**78.1**	15.9	6.0	**81.1**	13.0	5.9
2: Speech	0.9	**86.7**	12.5	1.7	**82.0**	16.2	2.3	**70.2**	27.5
3: Background	0.1	17.2	**82.5**	0.1	16.8	**83.1**	0.6	4.0	**95.4**

Table 5.19: Confusion Matrix obtained on training data by a RNN with 1 layer of 128 LSTM units, and a CNN with 5×15 filters with 10×10 max pooling (Configuration i, no post-processing).

	RNN			CNN		
DPR	34			195		
1: Shout	**96.2**	2.7	0.1	**80.9**	18.6	0.5
2: Speech	0.2	**96.8**	3.0	2.1	**79.4**	18.5
3: Background	0.0	2.4	**97.6**	0.2	18.8	**81.0**

Table 5.20: Confusion Matrix from a 1-layer RNN with 128 and 256 hidden units, on Configuration i, ii and iii (no post-processing applied).

layer size	128									256								
Config.	i			ii			iii			i			ii			iii		
1: Shout	**73.5**	21.4	5.2	**74.2**	23.2	2.6	**67.7**	24.8	7.5	**78.1**	15.9	6.0	**68.6**	27.3	4.1	**66.3**	24.8	8.9
2: Speech	0.9	**86.7**	12.5	1.4	**74.8**	23.8	1.2	**83.5**	15.3	1.7	**82.0**	16.2	1.4	**85.7**	12.9	0.7	**84.7**	14.6
3: Bg.	0.1	17.2	**82.5**	2.3	27.4	**70.4**	1.2	24.3	**74.6**	0.1	16.8	**83.1**	1.9	28.6	**69.5**	0.4	18.6	**81.0**

5.3.5 Streaming vs pre-Segmented

All the results presented so far were computed using pre-segmented data (Configuration i), meaning that all training and testing examples were composed of sequences of successive vectors where all vectors belong to the same class. As explained earlier, this way of processing the data is not realistic since real-time data is never pre-segmented. We present here the results obtained by using streaming data in Configuration ii and iii (see Section 5.2.8) and compare them with those obtained in Configuration i. We limit these experiments to one of the best NNs that we run in the previous reported experiments, namely the RNN with 1 layer of 256 LSTM units. The new results in Configuration ii and iii are reported in Table 5.20. Again, no post-processing is applied for the sake of comparison.

Configuration i vs ii. When comparing Configuration i with Configuration ii, results vary with the number of units. The RNN with 128 hidden units loses over 10% of accuracy for classes speech and background but maintain its score for class shout, while the RNN with 256 units loses 10% of accuracy for classes shout and background while slightly improving its score for class speech. So it seems that on the whole, testing the models with streaming data affects the classification accuracy but not so much since at least one class does not suffer at all in either case (shout for the RNN with 128 units and speech for the RNN with 256 units). So even though the results are definitely not as good in Configuration ii as they are in Configuration i, they remain at an interesting level of quality.

Configuration ii vs iii. When comparing Configuration ii with Configuration iii, results are a little more consistent between the two architectures. Class background benefits in both cases (by 4% for the RNN with 128 units and over 10% for the RNN with 256 units), class shout loses accuracy (by almost 7% for the RNN with 128 units and a little over 2% for the RNN with 256 units). Scores for class speech are improved by almost 9% for the RNN with 128 units, while they decrease by 1% for the RNN with 256 units. All in all, the overall accuracy (over the whole dataset) is better in Configuration iii, which makes perfect sense since the models are trained on streaming data, which is consistent with the test data.

Configuration iii vs i. Class shout loses 7% with the RNN with 128 units and almost 12% with the RNN with 256 units. Class speech loses barely 3% for the RNN with 128 units, while gaining amost the same amount for the RNN with 256 units. Class background loses about 8% for the RNN with 128 units and only 2% for the RNN with 256 units. Overall the RNN with 256 units performs pretty well in Configuration iii, maintaining scores over 80% for both class speech and background and scoring an accuracy of 66% on class shout. It seems that class shout is the one that is affected

the most by using streaming data, which is understandable since it the least represented one in the dataset.

5.4 Conclusion and discussion

In this chapter we tested different models (NN, CNN and RNN) for the 3-class task, classifying 'shout, 'speech' and 'background'. Using a NN, we first focused on the features to represent our data, using MFSC and MFCC with different settings, concluding that MFCC with a bandwidth of 50 to 20 kHz was the best choice. Then we tried several configurations of the NN architecture, varying the number of layers and number of neurons in each layer. The simplest configuration of 1×128 seemed to be the best choice,a s more complexity did not improve the result, suggesting that the amount of data was not enough to learn more complex networks. Moreover, the size of the context window used to provide temporal information had a big impact on the model's ability to detect the target patterns, with a bigger window of 50 frames providing more temporal context yielding better results. Generative pre-training of the NN did not improve the learning process, as discriminative training only was enough to find the global minimum of the error function. Using post-processing methods (smoothing and majority-voting) helped refine the model's predictions, bringing stability in consecutive outputs and preventing sparse changes in predictions. We concluded from Section 5.3.2 that the best NN uses 1 layer of 128 hidden units, taking 50-frames context window inputs, whose results are given in Table 5.8. In Section 5.3.3 we used the CNN model, to detect patterns across time and frequency by using 2-D filters to analyze input matrices representing the spectral evolution of the signal. Changing the filters' dimensions had some effect on the result, with filters of 5×15 having the best performance. Similarly to NN, a longer context window of 50 frames (concatenated together to form the input matrices) improved the model's ability to detect the target 'shout' and 'speech' classes, giving it more context to analyze patterns with a strong temporal structures, although losing accuracy on class 'background'. Finally in Section 5.3.4 we used RNNs to learn the temporal structure of patterns, with the aid of their memory pipe implemented through the LSTM units' gates. The high complexity of this model did not allow to test complex architecture with a lot of units and hidden layers, but the 1×256 configuration obtained the best results overall, as compared to the NN and CNN. The confusion matrices of the best models for each architecture are given in Table 5.21. Although the RNN didn't give the best recognition accuracy for class shout (78.1% vs 85.7% for CNN and 82.6% for NN) and class speech (82% vs 81.6% for CNN and 83.6% from NN), it performed better on class background (83.1% vs 77.7% for CNN and 71.4% for NN), giving the best overall performance, as seen in Table 5.22 (82.4% vs 80.5% for CNN ane 79.5% for RNN). Indeed, the RNN with LSTM units is able to perform slightly better overall, when taking into account the distribution of the data across our three classes in our test dataset, yielding a higher weighted average accuracy score.

By the look of the results obtained so far, the first conclusion is that all three models (NN, CNN and RNN) are not so far from one another, as shown in Table 5.21, but when considering the model's capacity on the whole task, Table 5.22 shows that the RNN is the best candidate for the task with 82.4% accuracy. Moreover, recall that by using 50-frames context window inputs, sound occurrences from all classes that were smaller than 50 frames were discarded from the test dataset. Therefore the NN and CNN did not have to classify the smaller events from each class, as opposed to the RNN which considered all events regardless of their length.

The prominence of RNN and especially with LSTM units in the litterature nudged our decision

Table 5.21: Confusion Matrix for the three best models (Configuration i, no post-processing); a DNN with 1 layer of 128 units, a CNN with 5×15 filters and 10×10 max-pooling plus 1 NN layer of 128 units, a RNN with 1 layer of 256 LSTM units

	1×128 NN			1×128 CNN			1×256 RNN		
1: Shout	**82.6**	14.1	3.3	**85.7**	12.2	2.0	**78.1**	15.9	6.0
2: Speech	1.0	**83.6**	15.4	1.9	**81.6**	16.5	1.7	**82.0**	16.2
3: Background	0.6	27.9	**71.4**	1.8	20.5	**77.7**	0.1	16.8	**83.1**

Table 5.22: Total accuracy weighted according to the class distribution in the test dataset, from the three-class results for the best NN, CNN and RNN models.

	Accuracy
NN	**79.5**
CNN	**80.5**
RNN/LSTM	**82.4**

towards keeping the RNN/LSTM, as we couldn't continue using both in further experiments for ressource availability reasons.

Although satisfactory, the results obtained in this section are not perfect and we believe could be improved. Given the environment considered in this study, we venture to assume that the major difficulty lies in the environment. We believe that the prominence of its content sometimes overshadows the content of the three speech-related classes. In the next chapter we set out to tackle this problem by taking on a new approach.

Chapter 6

Integrating contextual information from the acoustic environment: the 15-class task

In light of the discussion from Chapter 4, we believe that the results from Chapter 5 can be improved by considering the environment. In the present chapter we try to account for the noisiness and variability of the physical environment in which the target 'shout' signals occur. We explicitly categorize the surrounding environment/vehicle sounds into a set of classes, in addition to the 3 classes labeling defined in the Chapter 5 (shout / speech / background sounds). As detailed below, this results in a double-labeling into voice-related events and environment/vehicle-related events. The idea is to provide a finer description of the voice-related events (and thus the alert signals of interest), and to see if this finer description alleviates the possible burden of the classifier, compared to the raw 3-class problem, by reducing the variability of each class despite a higher number of classes.

6.1 Definition of environment/vehicle sound classes

To this aim, based on an extensive informal listening of the database, we defined the following 5 sound classes, each one globally corresponding to one specific phase of the train's motion and actions, as illustrated on the spectrogram in Figure 6.1:

- **Stand-by**: This class is defined as all situations where the train is not moving. When the train is operating on the line, the duration of this class usually stands between 15-20 s. When the train is parked in the workshop or at the end of the line, the duration ranges from 2 min (end of the line) to 10 min (workshop).
- **Compressor**: Specific noise made by the vehicle's compressors when about to start moving. Its duration is around 1 s. This type of noise has a spectral content ranging from 2000 Hz to 22050 Hz, with diminishing energy after about 600 ms.
- **Departure**: When the train engages in the process of starting its engine and moving forward, between the compressor noise and the moment it enters the tunnel. An energy stain specific to this event can be observed, beginning around 2000 Hz and progressing towards 5000 Hz at the end. This sound is very distinct to the ear.
- **Cruise**: Time span between two stations, when the train is inside the tunnel. The duration of this portion varies from 40 s to 90 s, depending on the distance between stations. Its spectral content varies according to the speed. In terms of spectrum representation, the speed leads to spectral lines that can be seen around 6000-7000 Hz in Figure 6.1. The stages of acceleration and deceleration can be identified, with a transition at around 26 s. Moreover this event is composed of very energetic noise along the entire frequency axis, stemming from the mechanical rolling.
- **Arrival**: occurs between the moment the train comes out of the tunnel and when it stops. As with the Departure event, it is easily identifiable upon listening. Its duration depends mainly on the length of the station.

As seen in Chapter 4 the acoustic environment evolves during the course of the metro's journey. In the following, for simplicity, these 5 classes of environment/vehicle noise are referred to as the 'environment classes'. In the example shown in Figure 6.1, some additional sounds can be identified such as; speech signal (between 0 s and 5 s), sound of crumpled newspaper right underneath the microphones (between 10 s and 16 s), internal and external mechanical noises due to railroad switch (between 18 s and 20 s) or due to the junction between two rails (between 34 s and 36 s).

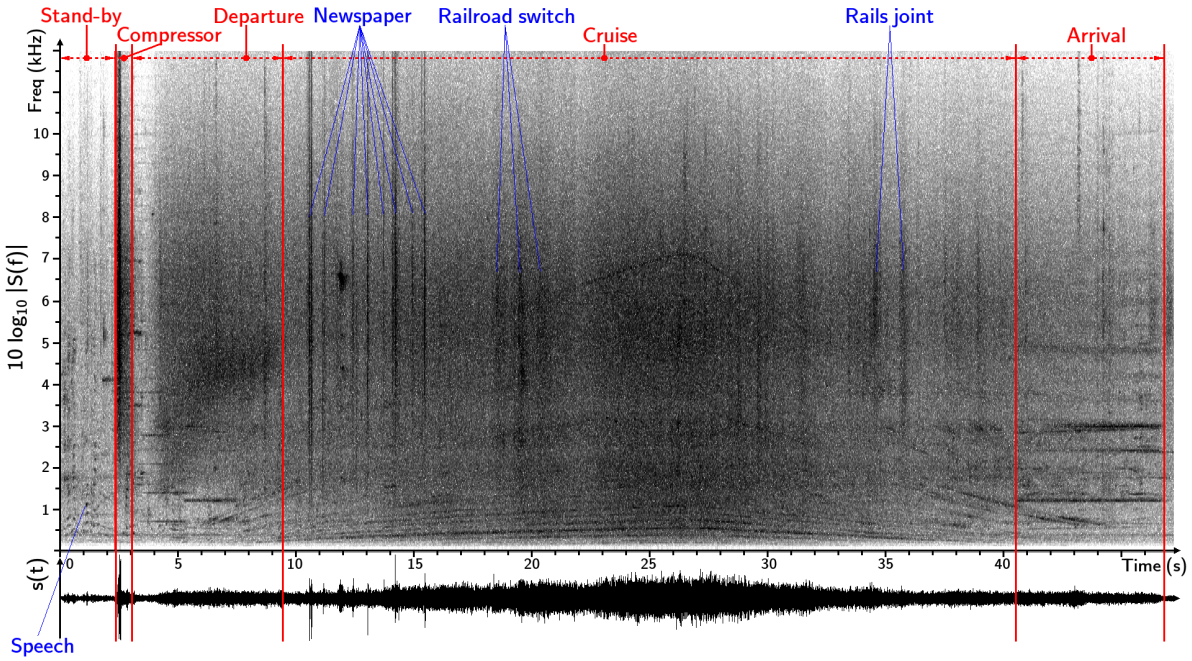

Figure 6.1: Example spectrogram of a recording while the train is traveling from one subway station to the next. The red lines show the boundary of the 5 environment-related classes. In blue are noted some examples of additional sound events.

6.2 Definition of the composite classes

The first solution that is adopted to provide context awareness (here vehicle noise) is to construct new labels by combining these newly created environment labels with the voice-related labels defined and used in Chapter 5 as an exhaustive 'product' list. This results in splitting the voice-related classes in subclasses according to the environment. For example, class 'shout' merged with the five environment classes gives the following five new classes : 'shout+stand-by', 'shout+compressor', 'shout+departure', 'shout+cruise' and 'shout+arrival'. The same thing goes for class 'speech' and class 'background'. Each of the three voice-related classes is thus combined with each of the 5 environment sound classes (or conversely the combination of each of the 5 environment classes with each of the 3 voice-related classes), to perform classification on $3 \times 5 = 15$ classes. Also, recall that class 'background' defined in Chapter 5 contains all 'non-vocal' noises (e.g. newspaper crumpling). Therefore, the combination of class 'background' with one of the 5 environment noise is to be seen as 'no shout and no speech' in the corresponding train movement/action phase.

For simplicity, this second approach with compound classes is referred to as the '15-class problem' (as opposed to the 3-class problem of Chapter 5). If $c(e)$ denotes the class composed of the voice label c and the environment label e, the classification task at time n now amounts to the following:

$$\widehat{c(e)}_n = \arg\max_{c(e)}(\mathbf{y}_n), \tag{6.1}$$

where $\mathbf{y}_n$ is the output vector composed of the estimated posterior probability of the 15 compound classes $y_{c(e),n}$. The voice-related estimated class $\widehat{c}$ is identified by the voice-related label associated with the $\widehat{c(e)}$ as the network's final prediction. Additionally, we identify the estimated environment class $\widehat{e}$ associated with $\widehat{c(e)}$ as the network's prediction regarding the environment. In short, we have $\widehat{c}(\widehat{e}) = \widehat{c(e)}$.

6.3 Experimental settings

6.3.1 Data and features

The data used in these new experiments are the same as in Chapter 5 (Sections 5.2.1 and 5.2.3). As for most experiments in Chapter 5, the input vectors are vectors of 40-MFCC coefficients, calculated every $T_i = 10$ ms, over a $T_w = 25$ ms time window.

As in Chapter 5 we split the database in subsets of training, validation and testing data. Table 6.1 presents the distribution of the data in seconds for these subsets. More specifically, this table provides the total number of seconds of data from voice-related class c (in rows) in environment-related class e (in column). In this 15-classes problem, the imbalance problem is more annoying than it was in the 3-class problem: some classes in different contexts only present very few occurrences such as 'shout+departure' and 'shout+arrival'. The compressor environment class is a small with few events, and even none for class 'shout+compressor'; due to the short duration of compressor action, scream/shout examples never happened to concur with compressor noise.

6.3.2 Networks settings

In Chapter 5, we observed that, globally, RNNs with LSTM units were the best model for our alert sound classification problem in the 3-class configuration. Since we want to limit the number of experiments, in these new experiments we rely on the RNN architecture with LSTM units. Two RNN architectures were tested, one is composed of 1 hidden layer of 128 units and the other one has 1 hidden layer of 256 units.

6.3.3 Training/testing settings

As before, weight optimization was done via SGD through the update equation (5.3), with cross-entropy as the likelihood function, as defined in Equation (3.19), and no momentum was used.

Out of the three different configurations introduced in Section 2.1.2 in Chapter 5, two were tested: Configuration i allows to evaluate the models on pre-segmented data. In Configuration ii the networks are tested on streaming data. Recall that in these two configrations the models are learned with pre-segmented data.

Table 6.1: Duration (in seconds) of the 15 composite classes in the train/validation/test datasets.

	Stand-by	Departure	Compressor	Cruise	Arrival	Total
Shout	65/1/13	6/0/2	-	228/41/14	7/1/4	358/43/30
Speech	650/68/187	124/8/40	19/3/6	472/57/105	91/24/25	1554/152/356
Background	290/36/57	100/28/22	19/4/4	663/115/88	69/11/9	1292/190/194

The experiments presented in the following were conducted with and without the post-processing algorithms defined in Section 2.1.8.

6.3.4 Raw and marginalized confusion matrices

As before the results below are given in the form of confusion matrices as introduced in Section 2.1.9. To allow for a comparison with previous results obtained for the 3-class problem (shout / speech / background sounds), and because the shout detection is the primary goal of the study, the result of these new 15-class experiments are also marginalized over the 5 environment sound classes, so as to end up with a 3-class classification result, over the three 'main' classes. We provide the 15-class confusion matrix and the corresponding 3-class matrix averaged over the 5 environment classes. To take into account the data distribution in this averaging process, we weight the classification scores for composite classes by the actual proportion they represent within the main classes. Formally, let $N_{c(e)}$ denote the total number of tested data frames of voice-related class c in environment-related class e (corresponding values in seconds are given in Table 6.1). Let $m_{c(e),g(l)}$ denote the percentage of occurrences of voice-related class c in environment sound class e classified as voice-related class g in environment sound class l. Then, the entries of the averaged 3-class confusion matrix, i.e. the percentage of occurrences of voice-related class c classified as voice-related class g, are given by:

$$\hat{m}_{c,g} = \frac{\sum_e \sum_l N_{c(e)} m_{c(e),g(l)}}{\sum_e N_{c(e)}}. \tag{6.2}$$

Additionaly, to evaluate the effect of the 15-class paradigm on the classification of environment classes, we also marginalized the results of 15-class problem over the 3 voice-related classes. Along the same lines as before, we estimated the entries of the averaged 5-class environment confusion matrix from the 15-class confusion matrix with the following equation :

$$\hat{m}_{e,l} = \frac{\sum_c \sum_g N_{c(e)} m_{c(e),g(l)}}{\sum_c N_{c(e)}}. \tag{6.3}$$

where $\hat{m}_{e,l}$ is the percentage of occurrences of environment-related class e classified as environment-related class l. we compared the results with the ones obtained with a RNN network of 1×256 LSTM units trained on the 5 marginalized environment classes (marginalized over the voice-related events).

For a consistent comparaison, the 5-class experiments are also conducted in Configuration i and ii, with majority vote post-processing.

Table 6.2: Confusion matrix of an RNN with 1×128 LSTM units for the 15 class problem in Configuration i (no post-processing).

	1(1)	1(2)	1(3)	1(4)	1(5)	2(1)	2(2)	2(3)	2(4)	2(5)	3(1)	3(2)	3(3)	3(4)	3(5)
1(1): Shout + Stand-by	4.2	0.1	0.2	**72.3**	0.0	9.0	1.9	0.2	2.5	5.2	2.1	0.2	0.4	0.2	1.4
1(2): Shout + Departure	0.0	0.0	0.0	21.8	0.0	20.8	3.9	0.0	1.4	34.2	0.0	4.9	0.0	2.4	10.6
1(3): Shout + Comp.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1(4): Shout + Cruise	0.0	0.0	0.0	**78.8**	0.0	0.0	0.0	0.0	10.8	2.4	0.1	0.1	0.0	7.0	0.9
1(5): Shout + Arrival	0.0	0.0	0.0	28.9	0.0	0.0	0.0	0.0	0.7	**52.9**	0.0	0.5	0.0	3.6	13.4
2(1): Speech + Stand-by	0.2	0.0	0.0	1.5	0.0	**75.2**	2.0	0.1	8.2	2.1	5.0	0.2	0.1	4.6	0.7
2(2): Speech + Departure	0.0	0.0	0.0	0.3	0.0	3.1	**47.9**	0.1	9.4	4.3	0.8	23.9	0.0	6.3	3.8
2(3): Speech + Comp.	0.0	0.0	0.0	0.2	0.0	0.6	1.9	6.6	0.0	0.0	1.6	8.4	**80.6**	0.0	0.0
2(4): Speech + Cruise	0.0	0.0	0.0	0.9	0.0	0.0	0.3	0.0	**59.0**	0.4	0.0	0.4	0.0	37.9	1.0
2(5): Speech + Arrival	0.04	0.0	0.0	0.6	0.0	3.6	2.0	0.0	3.4	**67.3**	0.0	0.0	0.0	1.4	21.5
3(1): Background + Stand-by	0.0	0.0	0.0	0.0	0.0	4.8	0.2	0.1	0.2	0.1	**74.4**	2.3	1.0	11.7	5.2
3(2): Background + Departure	0.0	0.0	0.0	1.0	0.0	0.5	1.6	0.0	7.4	0.0	3.1	**65.7**	0.1	6.0	14.6
3(3): Background + Comp.	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.7	0.0	0.0	4.5	7.0	**87.5**	0.0	0.0
3(4): Background + Cruise	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.0	7.4	0.1	0.4	0.0	0.0	**90.3**	1.4
3(5): Background + Arrival	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.6	0.8	0.0	0.0	1.8	**96.5**

6.4 Results

We first present the classification results for the 15 compound classes, before marginalizing over the environment-related classes in order to evaluate the global classification scores of each voice-related class with that method. We will compare the marginalized results with those from Section 5.3 to see if providing context awareness is helpful to the classifier. For concision in the presentation of tables, the classes names were replaced with numbers and $i(j)$ designates the voice-related class i (1: shout, 2: speech, 3: background) in environment-related class j (1: stand-by, 2: departure, 3: compressor, 4: cruise, 5: arrival). Finally we present the dual results for the environment classification (marginalized over voice-related events).

The effect of the majority vote post-processing being found to be beneficial with a gain of between 3% and 5% overall, we present only the results with post-processing.

6.4.1 Classification of compound classes

This section presents the results for the classification of the 15 composite classes in Configuration i and ii. Note that one class ('shout+compressor') is completely absent from those tests due to the absence of occurrences from the test set as mentioned in Section 6.3.1. Results are given for RNNs with 1 layer of 128 and 256 LSTM units. The results in Configuration i (network trained and tested with pre-segmented data) are given in Table 6.2 for the 128-unit RNN and in Table 6.3 for the 256-unit LSTM, and in Configuration ii in Table 6.4 for the 128-unit RNN and in Table 6.5 for the 256-unit LSTM.

RNN with 128 LSTM units, Configuration i. From Table 6.2 if we consider the 'shout' classes (labels 1(1) to 1(5)), class 'shout+cruise' (1(4)) is the only accurately classified with 78.8% accuracy. Moreover, most errors for the other 'shout' classes stem the confusion with this class. This makes sense when considering Table 6.1, as shouts occur mainly in 'cruise' environment and are scarce in other environment classes. Therefore the 'shout+cruise' class is the class that best represents all types of 'shouts' occurrences (in other words, the environment classes does not give improvement in this case). This indicates that the classifier does not have enough information

(maybe due to the lack of data) to model the spectral content outside the harmonics generated by shouts, in order to compensate the strong energy of those harmonic. In the case of 'shout+stand-by' (1(1)) the high confusion with 'shout+cruise' (72.3%) could be explained by the saturation sometimes present in the data. Indeed in that case all frequency content is masked by the saturation and therefore the classifier can not differentiate between 'stand-by' and 'cruise' environments, which have very different frequency contents and shoud not be confused by the classifier. Class 'shout+departure' suffers from the lack of representation (only 6 s in the training set and 2 s in the test set) and gets confused partly with 'speech+standby' (20.8%) and 'speech+arrival' (34.2%) by the classifier. Similarly, class 'shout+arrival' (28.9% of confusion with 'shout+cruise') suffers from a lack of representation (only 4 s in the training set and 7 s int the test set) and gets confused with 'speech+arrival' (52.9%) which indicates that the 'arrival' content takes over the 'shout' content. With relatively little data, the system recognizes the environment-related patterns inside the data but not the speech-related ones. Overall the shout classes are mostly confused with each other, therefore not creating much confusion with any 'speech' or 'background' compound classes (except for classes 'shout+departure' and 'shout+arrival' as mentioned).

As for 'speech' compound classes, the most represented one, 'speech+stand-by' (2(1)), gets 75.2% accuracy. For classes 'speech+departure' (2(2)), 'speech+cruise' (2(4)) and 'speech+arrival' (2(5)) the main confusion comes from their corresponding background compound class (between 21.5% and 37.9%), meaning that the environment aspect of those classes sometimes overshadows the speech aspect (again, the system recognizes the environment-related patterns inside the data but not the speech-related ones). This issue is even more pronounced for class 'speech+compressor' (2(3)) which gets mistaken for 'background+compressor' by 80.6%, because of its under-representation (19 s for training and 6 s for testing).

All 'background' compound classes get good to excellent accuracy (from 65.7% to 96.5%) showing the classifier's ability to recognize the environment in the absence of voiced content. The lowest accuracy being for class 'background+departure' (3(2)) with 65.7%, which is mainly confused with 'background+arrival' (3(5)) (14.6% of confusion) and not much with any speech-related class.

Table 6.3: Confusion matrix of an RNN with 1×256 LSTM units for the 15 class problem in Configuration i (no post-processing).

	1(1)	1(2)	1(3)	1(4)	1(5)	2(1)	2(2)	2(3)	2(4)	2(5)	3(1)	3(2)	3(3)	3(4)	3(5)
1(1): Shout + Stand-by	5.9	0.9	0.0	**66.2**	1.2	5.6	4.4	0.9	5.2	6.8	2.2	0.0	0.3	0.0	0.2
1(2): Shout + Departure	0.0	28.9	0.0	11.9	0.0	9.1	12.6	0.0	1.6	27.0	0.0	1.0	0.0	0.7	7.0
1(3): Shout + Comp.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1(4): Shout + Cruise	0.0	0.0	0.0	**73.0**	0.0	0.0	0.3	0.0	11.4	3.3	0.2	0.2	0.0	10.0	1.6
1(5): Shout + Arrival	0.0	0.0	0.0	19.6	0.0	0.0	4.1	0.0	7.2	**62.0**	0.0	0.0	0.5	0.0	6.7
2(1): Speech + Stand-by	0.2	0.0	0.0	2.2	0.0	**65.1**	2.3	0.2	9.7	4.1	14.1	0.4	0.0	1.2	0.5
2(2): Speech + Departure	0.0	0.0	0.0	2.5	0.0	2.1	**61.1**	0.3	6.8	1.0	0.4	17.7	0.1	2.0	6.1
2(3): Speech + Comp.	0.0	0.0	0.0	0.0	0.0	0.2	9.2	11.5	0.0	0.0	5.5	1.3	**72.2**	0.2	0.0
2(4): Speech + Cruise	0.0	0.0	0.0	0.7	0.0	0.4	0.2	0.0	**74.5**	0.8	0.1	0.5	0.0	21.9	0.8
2(5): Speech + Arrival	0.0	0.0	0.0	0.2	0.0	1.3	2.4	0.0	3.5	**68.7**	0.3	0.0	0.0	1.6	22.0
3(1): Background + Stand-by	0.0	0.0	0.0	0.0	0.0	2.3	1.5	0.2	0.3	0.9	**78.0**	1.4	1.5	9.0	4.9
3(2): Background + Departure	0.0	0.0	0.0	0.4	0.0	0.0	28.6	0.1	1.0	1.0	3.8	**51.0**	0.5	2.4	11.1
3(3): Background + Comp.	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.2	0.7	**91.2**	0.0	0.0
3(4): Background + Cruise	0.0	0.0	0.0	0.7	0.0	0.2	0.0	0.0	14.0	0.1	0.1	0.2	0.0	**83.0**	1.7
3(5): Background + Arrival	0.2	0.0	0.0	0.3	0.0	0.5	0.5	0.0	0.1	11.4	0.3	0.0	0.0	0.3	**86.5**

RNN with 256 LSTM units, Configuration i. From Table 6.3, the main tendency with a 256 units RNN is similar as with the 128 units RNN, with some improvements. Class 'shout+stand-by' gets less confused with class 'shout+cruise' (66.2%). Class 'shout+departure' jumps from 0% accuracy in the 128-unit case to 28.9%, which may be due to a better initialization since the amount of training data is still very low. Class 'shout+arrival' gets confused with 'shout+cruise' less than previously (19.6% vs 28.9%) but a little more with 'speech+arrival' (62% vs 52.9% previously). For 'speech' classes, the same trend as in the 128-case is seen, with nonetheless some improvements in accuracy for classes 'speech+departure' (+13.2%), 'speech+cruise' (+15.5%) and 'speech+arrival' (+1.4%). 'speech+stand-by' is the only class seeing its score decrease (−10.1%), with more confusion with class 'background+stand-by' (14.1% vs 5.02% previously) which shows that the model favored the environment patterns in the data more than the 128-unit for this class. Class 'speech+compressor' now gets more correct detection with a score of 11.5%, but still gets highly confused with 'background+comp' (72.2%). Overall there is a noticeable confusion with the corresponding 'background' classes, with respectively in the order of classes 14.1%, 17.7%, 72.2%, 21.9% and 22%

Strangely some 'background' classes loose a bit of accuracy with the 256-unit model compared to the 128-unit one (up to 14.7% losses), except for class 'background+stand-by' (+3.6%) and 'background+compressor (+3.7%), but only to a mild extent. Confusion with the corresponding 'speech' classes is high for classes 'background+departure', 'background+cruise' and 'background+arrival' (28.6%, 14% and 11.4% respectively). Overall however these scores show that the model is able to discriminate well between the different environments in the absence of voiced signals, as previously.

On the whole we can see that the environment doesn't really help the model in the 3-class task. Events from class 'shout' have a high detection rate and even overshadow the environment, leading the classifier to confuse all 'shout'-related classes. Finally, without presence of voiced signal all environments are detected pretty well.

Table 6.4: Confusion matrix of an RNN with 1×128 LSTM units in Configuration ii (no post-processing).

	1(1)	1(2)	1(3)	1(4)	1(5)	2(1)	2(2)	2(3)	2(4)	2(5)	3(1)	3(2)	3(3)	3(4)	3(5)
1(1): Shout + Stand-by	2.8	0.0	0.0	**76.1**	0.0	7.0	5.3	0.0	4.7	3.1	0.0	0.0	0.0	0.0	0.9
1(2): Shout + Departure	0.0	7.5	0.0	25.0	0.0	23.5	16.9	0.7	10.1	5.0	0.0	0.5	0.0	1.0	9.6
1(3): Shout + Comp.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1(4): Shout + Cruise	0.0	0.0	0.0	**79.9**	0.0	0.07	0.7	0.0	11.8	0.0	0.0	0.0	0.1	7.3	0.1
1(5): Shout + Arrival	0.0	0.0	0.0	41.0	0.0	0.3	0.77	0.0	1.3	**55.1**	0.0	0.0	0.0	0.0	1.5
2(1): Speech + Stand-by	0.1	0.0	0.0	4.2	0.0	**68.8**	2.8	0.2	13.1	2.9	4.4	0.1	0.1	2.7	0.5
2(2): Speech + Departure	0.0	0.0	0.1	3.5	0.0	5.0	**61.1**	0.4	3.7	2.1	1.1	12.2	0.54	6.2	4.1
2(3): Speech + Comp.	0.7	0.0	0.0	2.5	0.0	**40.3**	8.3	11.0	0.0	0.0	0.3	0.0	36.8	0.0	0.0
2(4): Speech + Cruise	0.0	0.0	0.0	3.4	0.0	0.01	1.4	0.0	**66.2**	0.4	0.0	0.3	0.0	27.3	0.9
2(5): Speech + Arrival	0.0	0.0	0.0	0.8	0.0	0.0	3.1	0.0	4.7	**70.2**	0.0	0.0	0.0	0.4	20.8
3(1): Background + Stand-by	0.2	0.0	0.0	0.4	0.0	20.6	1.8	0.0	4.7	0.7	**59.2**	0.5	1.5	4.4	5.8
3(2): Background + Departure	0.1	0.0	0.0	1.2	0.0	1.3	27.3	0.5	0.7	0.6	5.8	**49.5**	1.5	1.0	10.5
3(3): Background + Comp.	0.0	0.0	0.0	0.0	0.0	**46.5**	0.4	0.2	12.8	0.0	18.6	0.0	21.6	0.0	0.0
3(4): Background + Cruise	0.0	0.0	0.0	10.2	0.0	0.0	0.3	0.0	34.7	0.1	0.0	0.3	0.0	**52.4**	2.0
3(5): Background + Arrival	0.0	0.0	0.0	1.5	0.0	1.5	0.7	0.0	2.0	**46.2**	1.7	0.8	0.0	0.6	44.9

RNN with 128 LSTM units, Configuration ii. Looking at Table 6.4, all 'shout' classes seem to be treated in a very similar way as in Configuration i, with a 1.1% decrease for the main one ('shout+cruise'). However it seems that the model had a higher tendency of classifying the smaller classes into the most prominent one ('shout+cruise') (about $+4\%$ for 'shout+stand-by', $+3\%$ for 'shout+departure' and $+12\%$ for 'shout+arrival'). 'speech' classes also get a very similar treatment as in before, but surprisingly some of them are more accurately detected. Indeed, although 'speech+stand-by' looses about 6%, 'speech+departure' improves by 13%, 'speech+cruise' by 7% and 'speech+arrival' by 3%). Again, the confusion happen with the corresponding 'background classes' (between 12.2% and 36.8%) except for 'speech+stand-by' which gets confused with 'speech+cruise' by 13.1%. Interestingly now, most confusion for class 'speech+comp' comes from 'speech+stand-by' (almost 50%) instead of the corresponding background class, i.e 'background+comp'. This stems from the fact that the test data is no longer segmented according to the classes in Configuration ii, meaning short events such as compressor sounds are mixed with other neighboring events. The model does not have the precision or agility to be able to distinguish the short events frames from the rest at test time. Thus the confusion does not come from other similar classes (such as 'background+comp') but from classes surrounding the short events. 'background' classes are much more impacted than the rest by the streaming data at test time, drastically losing detection accuracy. Compared to the 128 unit RNN in Configuration i from Table 6.3, 'background+stand-by' loses about 15%, 'bacground+departure' 16%, 'background+comp' more than 60%, 'background+cruise' almost 40% and 'background+arrival' over 50%. The confusion brought to the classes most impacted by Configuration ii happens with the corresponding 'environment' class, meaning that the model is less capable of identifying the speech-related patterns in the signal than in Configuration i, and favors the environment-related patterns.

Table 6.5: Confusion matrix of an RNN with 1×256 LSTM units in Configuration ii (no post-processing).

	1(1)	1(2)	1(3)	1(4)	1(5)	2(1)	2(2)	2(3)	2(4)	2(5)	3(1)	3(2)	3(3)	3(4)	3(5)
1(1): Shout + Stand-by	2.2	0.0	0.4	**68.2**	0.8	8.4	3	0.1	5.2	9.5	0.2	0.0	0.0	0.0	1.9
1(2): Shout + Departure	0.0	2.1	0.0	15.9	0.0	36.3	1.5	0.0	2.5	25.5	0.7	1.1	0.0	1.0	13.1
1(3): Shout + Comp.	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
1(4): Shout + Cruise	0.0	0.0	0.0	**77.4**	0.0	0.0	0.0	0.0	9.8	0.7	0.0	0.2	0.1	9.4	2.4
1(5): Shout + Arrival	1.0	0.5	0.3	12.6	0.0	0.0	1.0	0.0	2.6	**76.9**	0.0	0.3	0.0	1.0	3.8
2(1): Speech + Stand-by	0.2	0.0	0.0	2.0	0.0	**69.3**	2.5	0.0	10.3	3.6	7.9	0.1	0.1	2.9	1.0
2(2): Speech + Departure	0.0	0.0	0.0	0.3	0.0	0.9	**47.9**	0.2	6.6	7.3	0.6	18.9	0.5	10.2	6.6
2(3): Speech + Comp.	0.0	0.0	1.5	0.2	0.0	5.5	9.5	17.2	21.3	0.0	4.5	1.3	**38.8**	0.2	0.0
2(4): Speech + Cruise	0.0	0.0	0.0	1.6	0.0	0.1	1.8	0.0	44.8	1.8	0.1	1.5	0.0	**44.9**	3.4
2(5): Speech + Arrival	0.0	0.0	0.0	0.3	0.0	1.9	4.9	0.0	5.9	**57.8**	0.2	0.2	0.0	0.3	28.5
3(1): Background + Stand-by	0.3	0.0	0.0	0.0	0.0	13.8	2.3	0.3	5.7	0.15	**64.1**	0.5	1.5	4.4	6.9
3(2): Background + Departure	0.9	0.0	0.1	0.5	0.0	0.6	11.6	0.1	1.3	1.2	3.1	**63.7**	0.8	2.2	13.8
3(3): Background + Comp.	3.6	0.0	2.0	0.0	0.0	3.2	4.6	1.0	**42.9**	0.0	16.0	0.2	26.1	0.4	0.0
3(4): Background + Cruise	0.0	0.0	0.0	5.9	0.0	0.1	0.2	0.0	15.7	0.5	0.1	0.9	0.0	**73.6**	3.1
3(5): Background + Arrival	0.0	0.0	0.0	0.1	0.0	0.3	0.1	0.0	0.4	36.0	4.1	0.8	0.0	1.0	**57.2**

RNN with 256 LSTM units, Configuration ii. According to Table 6.5, for the 256-unit RNN, the over-representation of class 'shout+cruise' is still very visible, with 77.4% acuracy. Class 'shout+arrival' is the only one which gets more confusion outside the 'shout' classes, with 'speech+arrival' by 76.9%, i.e 15% more than in Configuration i. Speech classes are treated in a very similar way as in Configuration i (with variations between 5% and 12%). Overall the confusion with the corresponding 'background' classes is still high, even higher than in the 128-unit cases, with respectively in the order of classes 7.89%, 18.9%, 38.8%, 44.9% and 28.5%. However, 'speech+compressor' now gets a lot less confusion with 'background+compressor' (38.8% vs 72.2%) and more with 'speech+cruise' (21.3% vs 0%).

For 'background' classes, 'background+stand-by' and 'background+cruise' had slight decrease in accuracy compared to Configuration i (respectively 14% and 9%). 'background+departure' has an increase of 12% and gets less confusion with 'speech+departure'm than in Configuration i (11.6% vs 28.6%) and more confusion with other 'background' classes. 'background+comp' loses almost 60%, again because the events are short. Similarly, 'background+arrival' loses up to 30%, getting confused with 'speech+arrival' mostly (36%). Confusion with the corresponding 'speech' classes is still high for classes 'background+departure', 'background+cruise' and 'background+arrival' with respectively 11.6%, 15.7% and 36%, and also for class 'background+stand-by' with 13.8%.

Overall Configuration ii has the most impact over the detection of classes 'speech+departure', 'speech+cruise', 'speech+arrival', 'background+stand-by' 'background+comp', 'background+cruise', 'background+arrival' with over 10% of loss in accuracy. The rest of the results are rather similar, and surprisingly class 'background+departure' has an increase of 12.7% .

6.4.2 Marginalizing

The averaged results (averaged across environment-related sound classes, see Equation (6.2) are shown in Table 6.6(left) for Configuration i, and in Table 6.6(right) for Configuration ii.

RNN with 128 LSTM units Comparing Table 6.6(left) with Table 5.18 for the RNN with 128 LSTM units, it seems that the 15-class task is harder than the 3-class task, as accuracy decreases

Table 6.6: Confusion matrix of 3 voice-related classes stemming from the 15-class confusion matrix marginalised over 5 environment labels, in Configuration i and ii for a RNN with 128 LSTM units.

context	Config i			Config ii		
1: Shout	**70.7**	23.1	6.2	**73.7**	22.2	4.1
2: Speech	1.1	**75.9**	23	3.9	**79.7**	16.4
3: Background	0.3	6.0	**92.3**	5.4	33.9	**60.7**

Table 6.7: Confusion matrix of 3 voice-related classes stemming from the 15-class confusion matrix marginalised over 5 environment labels, in Configuration i and ii for a RNN with 256 LSTM units.

context	Config i			Config ii		
1: Shout	**81.0**	12.3	6.7	**52.4**	35.4	12.2
2: Speech	1.4	**80.5**	18.1	4.3	**76.5**	19.2
3: Background	0.7	10.8	**88.5**	2.7	23.6	**73.7**

for class shout and class speech (by 3.5% and 10% respectively), even though it increases by 10% for class background. Curiously Configuration ii (Table 6.6(right)) seems to help the model (in the 15-class task) for class shout and class speech, with accuracy gains of 3% and 4% respectively, although it drastically impairs detection of class background. The improvement over classes 'shout' is due to the fact that the most prominent class ('shout+scruise' and 'speech+cruise') overshadowed the smaller ones even more in Configuration ii than in Configuration i, as explained in Section 6.4.1. For 'speech' classes this shows that the accuracy lost actually came from confusion with other 'speech' classes, so when marginalizing over the environment classes the confusion over the environment for the 'speech' classes turn out to count towards correct detection of class speech. When compared with the results for the 3-class task in Configuration ii directly from Table 5.18 it turns out that the 15-class task is quite similar, performing better on class speech ($+5\%$), similarly on class shout (-1%) and a lot less on class background (-10%).

RNN with 256 LSTM units Comparing Table 6.7(left) with 1 layer of 256 units from Table 5.18 (raw 3-class problem) it appears that in Configuration i context awareness through composite classes bumped up detection accuracy by 3% for shouts and about 5% for background sounds, while slightly reducing it by 1.5% for the 'speech' class. In regards of the results in Table 6.3, most confusion for classes 'shout' and 'background' are with classes from their own respective voice-group. For the 'speech' class however, the system confuses 'speech' with 'background' even if it does not make error on the associated environment, which is a consquence from what the fact described earlier that the system recognizes the environment-related patterns inside the data but not the speech-related ones.

In Configuration ii however, the conclusion is starkly different, as accuracy decreased for all 3 classes, compared with the 3-class approach from Section 5. Directly addressing the 3-class problem seemed to be significantly more efficient, since the scores for all classes in Table 6.7(right) are lower than those in Table 5.20 for Configuration ii. This results is the direct consequence of the increased segmentation of sequences, as explained in Section 6.4.1.

6.4.3 Environment Classification

Table 6.8 shows the results of classification of environment-related classes with a raw 5-class environment in Configuration i (pre-segmented data for train and test phases). Table 6.9 presents for the Configuration i and ii the mariginalized results of Table 6.2 and Table 6.4 over the voice-related labels using equation 6.2.

RNN with 128 **LSTM units** To begin with, in a raw 5-class environment problem in Configuration i (Table 6.8(left)), the results are satisfying : classes 'stand-by', 'compressor', 'cruise' and 'arrival' present very good recognition accuracy with respectively 89.7%, 90%, 98.1% and 87.8%. However, class 'departure' was a little less performant with a score of 73.8%. Using the 15-class results in the same Configuration i we can observe (Table 6.9(left)) a loss in accuracy for class 'stand-by' by 12.9%, and slight losses for classes 'departure', 'compressor' and 'cruise' by respectively 2.9%, 2.3%, and 0.2%, while class 'arrival' gains 0.5%. All in all the performance is on par with the 5-class classification shown in Table 6.8(left) except for class 'stand-by'.

In Configuration ii, using the marginalized results from the 15-class task given in Table 6.9(right) have very similar scores than the direct 5-class Configuration (Table 6.8(right)) for classes 'departure' and 'cruise' (within 0.1% difference). Class 'compressor' , although its accuracy' increases by 13.4%, still gets a high confusion with class 'stand-by' (50.5%). Class 'stand-by' is less accurately detected (71.6% vs 86.5%) and conversely class 'arrival' has a higher accuracy (84.7% vs 71.4%).

As a conlusion, for the RNN with 128 units the results from the 5-class task are slightly better in Configuration i than those from the 15-class task marginalized, but less so in Configuration ii.

Table 6.8: Confusion matrix for the 5 environment-related classes (the raw 5-class problem) in Configuration i (left) an ii (right) for a RNN with 128 LSTM units

context	Config. i					Config. ii				
1: Stand-by	**89.7**	0.9	0.3	6.2	2.6	**86.5**	1.7	0.4	7.2	3.9
2: Departure	11.4	**69.8**	0.1	12.7	6.0	15.9	**72.2**	0.7	5.7	4.9
3: Comp.	2.9	0.0	**90.0**	0.2	2.2	**67.7**	1.0	24.3	4.3	2.3
4: Cruise	1.0	0.1	0.0	**98.1**	0.7	0.4	1.8	0.0	**97.1**	0.6
5: Arrival	1.1	6.4	0.2	3.6	**87.8**	2.7	9.9	0.0	16.0	**71.4**

RNN with 256 **LSTM units** Table 6.10 shows the results of classification of environment-related classes with a raw 5-class environment in Configuration i (pre-segmented data for train and test

Table 6.9: Confusion matrix of environment-related labels from the 15-class confusion matrix marginalised over 3 'voice' labels, in Configuration i and ii for a RNN with 128 LSTM units.

context	Config i					Config ii				
1: Stand-by	**76.8**	2.2	0.4	16.9	3.6	**71.6**	2.9	0.6	20.8	4.1
2: Departure	3.7	**67.9**	0.1	15.1	10.0	6.2	**72.2**	1.4	9.5	7.6
3: Comp.	3.3	8.5	**87.7**	0.3	0.1	**50.5**	5.2	37.7	6.6	0.0
4: Cruise	0.2	0.4	0.0	**97.9**	1.5	0.1	1.2	0.0	**97.2**	1.6
5: Arrival	2.6	1.4	0.0	7.7	**88.2**	1.1	3.2	0.0	11.0	**84.7**

Table 6.10: Confusion matrix of the 5 environment-related classes (the raw 5-class problem) in Configuration i (left) an ii (right) for a RNN with 256 LSTM units

context	Config i					Config ii				
1: Stand-by	**89.6**	2.1	0.4	4.6	3.3	**85.4**	3.0	0.2	8.4	3.0
2: Departure	11.6	**78.7**	0.5	5.3	3.9	17.6	**73.0**	2.6	3.2	3.6
3: Comp.	3.4	0.0	**93.4**	0.1	3.0	**61.5**	0.1	35.6	0.5	2.3
4: Cruise	1.6	1.2	0.1	**96.3**	0.8	0.8	3.5	0.0	**95.1**	0.5
5: Arrival	6.9	5.7	0.9	2.1	**84.4**	9.7	6.7	0.2	20.1	**63.4**

Table 6.11: Confusion matrix of environment-related labels from the 15-class confusion matrix marginalised over 3 'voice' labels, in Configuration i and ii for a RNN with 256 LSTM units.

context	Config i					Config ii				
1: Stand-by	**76.1**	2.9	0.6	15.2	5.0	**74.2**	2.7	0.5	17.1	5.5
2: Departure	2.9	**76.5**	0.5	8.4	8.5	2.5	**67.6**	0.8	12.2	13.8
3: Comp.	6.7	6.7	**86.5**	0.2	0.0	14.8	8.4	**46.5**	30.3	0.0
4: Cruise	0.4	0.5	0.0	**97.2**	1.9	0.2	2.2	0.0	**93.3**	4.3
5: Arrival	1.4	2.2	0.1	6.6	**89.8**	2.5	3.7	0.0	6.4	**87.3**

phases). Table 6.11 presents for the Configuration i and ii the mariginalized results of Table 6.3 and Table 6.5 over the voice-related labels using equation 6.2.

In Configuration i the 5-class model is better at detecting classes 'stand-by' (89.6% vs 76.1%) by quite a margin, 'compressor' (93.4% vs 86.5%) by a smaller but still noticeable margin, and 'departure' by a very small margin (78.7% vs 76.5%), slightly worse at detecting class 'cruise' (96.3% vs 97.2%) and a little more for class 'arrival' (84.4% vs 89.8%). All in all the environment classes are pretty well classified in both cases, and it is hard to differentiate between the 5-class and the 15-class tasks since the differences in scores are not significant except for class 'stand-by'. The latter suffered from the lack of 'shout' data, which created a lot of confusion with class 'cruise' as seen in the 15-class results given in Table 6.3, and confirmed by Tab 6.10. For class 'compressor' more confusion with classes 'stand-by' and 'departure' occur in the 15-class task, because class 'speech+compressor' has too little training occurrences to train a good model on it. Therefore, although the model is able to detect compressor noises in absence of voiced sounds (i.e class 'background+compressor' with a score of 90.3%) the poor score on class 'speech+compressor' dragged the overall recognition of class 'compressor'. On the other hand, class 'arrival' appears to be better explained when considering the voiced content; which brings down the confusion with classes 'stand-by' and 'departure'. However confusion with class 'cruise' is higher in the 15-class task because of the imbalance between 'shout' classes (with 'shout+cruise' being prominent as explained in Section 6.4.1). Class 'departure' also suffers from this issue, as confusion with class 'cruise' increases by 3%.

All in all in Configuration i the environment-related classes are still fairly well classified despite the imbalance of the 'shout' classes which created more false alarms from classes 'stand-by', 'departure' and 'arrival'.

In Configuration ii a similar effect is noticed on class 'stand-by'. For class 'departure' the decrease is more important from the 5-class task to the 15-class task (−5.4% from 73% to 67.6%) than it was in Configuration i (−2.2%), as more confusion arises with classes 'cruise' and 'arrival'

in the 15-class task ($+9\%$ and $+10\%$ respectively), even though confusion with class 'stand-by' is reduced by 15%. This is because in the 15-class case the model is not able to identify class 'speech+departure' (47.9% per Table 6.5), which is where most errors come from for class 'separture' when marginalzing since it represents more than 50% of the content of class 'departure' (40 s over a total of 64 s as shown in Table 6.1). Indeed this class gets a lot of confusion with classes 'speech+cruise', 'speech+arrival', 'background+cruise' and 'background+arrival', with a total confusion of 16.82% with the 'cruise' classes and 13.88% with the 'arrival' classes (Table 6.5). Class 'compressor' has a better score in the 15-class task, perhaps surprisingly, with 46.5% vs 35.6% in the 5-class task. All confusion with class 'stand-by' (which represented 61.5% of the errors in the 5-class task) is now down to 14.8%, although confusion with class 'cruise' rises from 0.5% to 30.3%. It seems that the information about the voice context scatters the error across classes (also 8.4% of confusion with 'departure') as compared to the 5-class task with all confusion concentrated on one class, but in doing so happens to help detection of class 'compressor'. Class 'cruise' gets a similar treatment in both 5-class and 15-class tasks, with 95.1% and 93.3% accuracy respectively. Class 'arrival' benefits from the voiced context information as confusion with class 'cruise' decreases from 20.1% to 6.4%, as it seems that the classifier was not able to distinguish much the voiced content in that class. Indeed class'shout+arrival', 'speech+arrival' and 'background+arrival' got confused with each other a lot according to Table 6.5.

6.5 Discussion

In this chapter, based on the analysis of the database provided in Chapter 4, we decide to take into account the environment by providing contextual information to the network. We used 5 new labels related to the environmental sound events which constitute the acoustic background to create compound classes with the 3 speech-related classes used in Chapter 3. We performed classification of the resulting 15 classes with a RNN with 1 layer of 128 units and another one with 1 layer with 256 units. We then marginalized the results of this classification over the environment classes to obtain classification scores over the 3 speech-related classes. We also marginalized over the speech classes to obtain classification scores over the 5 environment-related classes, in order to compare them with direct classification of those 5 classes by the same networks.

From the perspective of speech-related classes, it appears from the marginalized score for the 3 speech-related classes that in Configuration i providing information from a different context (the context of the environment of the vehicle here) improved the results from section 5.3 for a RNN with 256 LSTM units with better scores for class 'shout' ($+3\%$) and 'background' ($+5\%$) although slightly lowering accuracy for class 'speech' by 1.5%. The conlusion was different for a RNN with 128 units, with lower accuracy for class 'shout' (-3.5%) and 'speech' (-10%) even though accuracy for class 'background' was higher by 10%. 128 units probably did not allow enough modelling power to the classifier to handle the complexity of the task. In Configuration ii however, the contextual information brought by the 15-class task helped the model in detecting class 'shout' and 'speech' ($+3\%$ and $+4\%$ repectively) for the RNN with 128 LSTM units. For the 256-unit RNN accuracy for all classes decreased in the 15-class task.

The 15-class approach seemed to be more robust to the real-life conditions of Configuration ii for a RNN with 128 units, while it the opposite for the RNN with 256 units. These results are not enirely satisfactory since they do not allow us to draw a clear conclusion as to the usefullness of the 15-class approach. Although the compound classes were more accurate and easier to model because they took into account the complexity of the acoustic environment, the complexity added by the high number of classes compared to the 3-class task was certainly a hurdle. As a result the gain from the more accurate classes was balanced by the added complexity. In the next chapter we will try a different way of bringing contextual information from the environment, by adopting a new paradigm.

Chapter 7

Classification of voice and environment events in a multi-label setting

In our attempt to bring context awareness to the classifier in Chaper 7 we actually raised the complexity. In the following we explore a different way of providing context information to the system by introducing a multi-label paradigm, after presenting the problem and its theoretical foundations. This paradigm is suited for the classification of simultaneous sound sources, since it allows multiple labels at the same time. The classifier is given the ability to output multiple classes to recognize sound events occurring concurrently. These experiments, being at an early stage, were only conducted in configuration i as the goal here is simply to measure their performance against the other models presented in Chapter 5 and Chapter 6.

7.1 Multiple audio sources viewpoint

In our last approach, we defined each sound event by two labels, an environment-related label and a speech-related one. By merging those two labels into one, as explained in Section 6.2, we basically built a single-label paradigm in which each individual class represents a voice-related label realized in an environment-related label. To tackle this paradigm we use a single-label classifier to perform what can be viewed as a multi-label task problem [122] (i.e to train the model with and detect multiple events at the same time) since each sound event is characterized by two labels. Additionally, note that the two sets of labels (speech-related and environment-related) are by definition completely independent since one represents mechanical sounds and the other represents voiced sounds.

From a global perspective, the classes involved in our problem could fit in a multi-label framework by considering each voice-related class as follows: class 'speech' only designating speech, class 'shout' only designating shouts, and class 'background sound' replaced by the class of environment sounds (defining by the vehicle motion). By these assumptions, the classes would not be mutually exclusive and the task would be referred to as multi-label classification.

However, in our problem, the labels are built to detect a particular situation rather than a precise category of voiced sounds: Indeed recall from Section 5.1 that the voice-related classes are defined as follow: class 'shout' represents shouts are heard with and without other speech occurrence; class 'speech' represents all voiced sounds heard without shout, and class 'background' is defined as all other situations. These definitions imply that the three classes are mutually exclusive and represent different situations of mixed sounds. Therefore they imply a certain complexity: an occurrence of class 'shout' can contain speech and shout sounds and thus comprise different spectral and temporal characteristics, implying a complicated discrimination between classes 'speech' and 'shout'.

Given the afore-mentioned considerations, we set out here to exploit the environment labels as in Chapter 6 to better model the various realizations of the voice-related classes by using the multi-label framework. In this case, at each step n, a sound segment is defined as in Chapter 5 by a voice-related speech and independently by a second class from the environment-related classes. Therefore, as in Chapter 6, the voice label is associated with an environment label in order 'to explain' the possible difference between two observations from a same label (ex speech in stand by and speech in cruise). The main difference with Chapter 6 is the decrease in model complexity: the total number of classes is now defined by the sum of voice-related and environment-related classes, whereas in Chapter 6 the total number of classes was given by the product of those classes.

7.2 Multi-label network training

The multi-label classification being already presented in Section 2.1.3, we present in the following the main differences between a multi-label network and the single-label networks used in Chapters 5 and 6.

Firstly, recall that in a 'single-label' task where one class c is chosen among C classes, element y_c of the output vector $\mathbf{y}$ of a ANN network (NN,CNN,RNN) is computed as:

$$y_c = \frac{\exp -(\mathbf{w}_c \mathbf{h}^{(L)} + w_{c0})}{\sum_{i=1}^{C} \exp -(\mathbf{w}_i \mathbf{h}^{(L)} + w_{i0})} \tag{7.1}$$

where $\mathbf{h}^{(L)}$ is the vector of units from the last hidden layer and the vectors $\mathbf{w}_i$ and w_{i0} are the associated parameters. Therefore, the class $\hat{c}$ is estimated by:

$$\hat{c} = \arg\max_{c}(\mathbf{y}) \tag{7.2}$$

By definition (Section 2.1.3), the multi-label paradigm allows to detect zero, one or several classes in the network's output $\mathbf{y}$. This is implemented by replacing the sigmoid function with the softmax function in (7.1):

$$y_c = \frac{1}{1 + \exp -(\mathbf{w}_c \mathbf{h}^{(L)} + w_{c0})} \tag{7.3}$$

All elements y_c are independent and have values between 0 and 1. As already seen in Equation (2.5) the set of detected classes $\hat{\mathbf{c}}$ is composed of all classes c with corresponding y_c greater than a threshold γ:

$$\hat{\mathbf{c}} = \{c, \text{if } y_c > \gamma | c \in [1, C]\} \tag{7.4}$$

Therefore, the likelihood becomes, as defined in Equation (3.19) from Section 3.1.1:

$$p(\mathbf{t} \mid \mathbf{x}, \mathbf{w}) = \prod_{j=1}^{N} \prod_{i=1}^{C} p(t_i \mid \mathbf{x}, \mathbf{w_i}) = \prod_{j=1}^{N} \prod_{i=1}^{C} y(\mathbf{x}, \mathbf{w_i})^{t_i} (1 - y_i(\mathbf{x}, \mathbf{w}))^{(1-t_i)} \tag{7.5}$$

where $\mathbf{x}, \mathbf{w}, \mathbf{x}, \mathbf{t}$ are respectively the input data, the weight vector and the target vector. The term $(1 - y_i(\mathbf{x}, \mathbf{w}))^{(1-t_i)}$ penalizes potential units which would be on (indicating that the input belongs to their corresponding class) when they shoud not, therefore correcting false alarms. It wasn't necessary in the single-label case since the softmax function ensures that only one class is output at a time. Thus, in that case when the wrong class is output, the unit corresponding to the target class which should have been detected will have a low probability $y(\mathbf{x}, \mathbf{w_i})$ whereas its associated target variable t_i will have a value of 1, therefore being penalized by the term $y(\mathbf{x}, \mathbf{w_i})^{t_i}$. On the other hand, in the multi-label paradigm it could be the case that the wrong class is output at the same time as the correct one, in which case the term $y(\mathbf{x}, \mathbf{w_i})^{t_i}$ does not penalize the false detection. This is why the term $(1 - y_i(\mathbf{x}, \mathbf{w}))^{(1-t_i)}$ is needed to penalize all false detections. The gradient of this function turns out to be similar to that of cross-entropy (see [10] for more details).

7.2.1 Notes on multi-label evaluation

In the multi-label context, the number of potential errors that the sytem can make is more important than in a single-label context. If the total number of target classes is C then the number

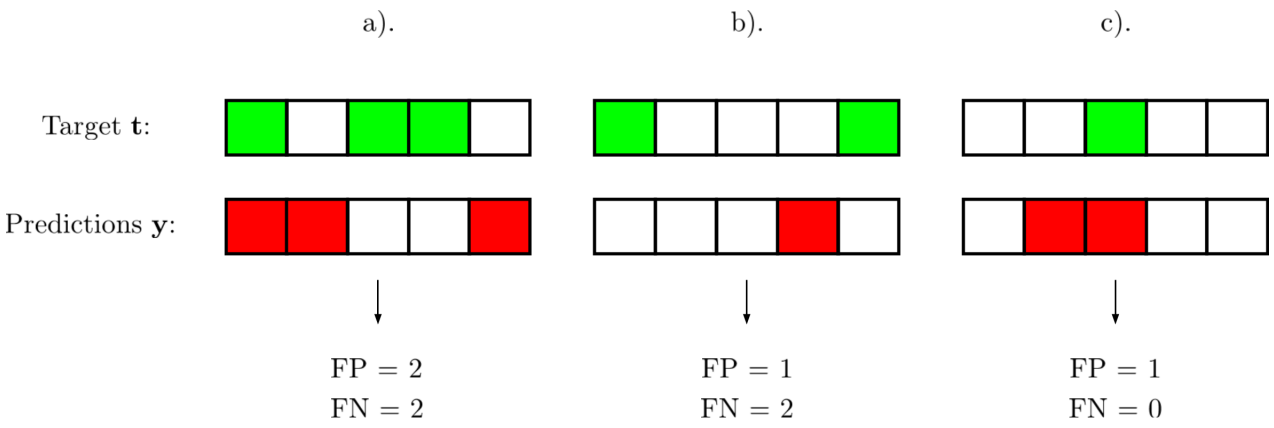

Figure 7.1: Three examples of multi-label target vectors and their Classification results in multi-label

of potential errors to be made for each input is C. The total number of possible errors over the whole training dataset is now $C \times N$ with N the number of inputs, while in a single-label context it was only possible to make as many errors as the number of data, i.e N. The example given in Figure 7.1 shows three target vectors (in green) and the associated predictions from the system (in red), showing the corresponding values of false positive (FP) and false negative (FN). If one error is counted for each class where the prediction does not match the label, the number of errors made by the classifier in Figure 7.1 is 4 for graph (a), 3 for (b) and 1 for (c).

Whereas in a single-label context the amount of false alarms were connected to the amount of accurate detection, since an accurate prediction equates no false alarms, here they are totally unrelated, and a prediction can carry accurate detections as well as false alarms at the same time. This is why the Recall, Accuracy and F-score metrics defined in Section 2.1.9 are well suited for multi-label experiments, because they give a better account of false alarms and accurate detection separately, as explained in Section 2.1.9.

7.3 Class definition in multi-label context

We consider the set of 3 voice-related classes as defined in Chapter 5 (mutually exclusive by definition). On the other hand we consider independently the 5 environment-related classes defined in Section 6.1 that are also mutually exclusive by definition (the environment cannot be in 'stand-by' state and in 'cruise' state at the same time). The total number of classes C is equal to $3 + 5 = 8$.

We note $\hat{\mathbf{c}}_n$ and $\hat{\mathbf{e}}_n$ respectively the set of voice-related class detected and the set of environment-related class detected at step n. Considering Equation (7.4), the decision for each set of class is made according to:

$$\hat{\mathbf{c}}_n = \{c, \text{if } y_{c,n} > \gamma | c \in [' background', ' speech', ' shout']\} \tag{7.6}$$

$$\hat{\mathbf{e}}_n = \{c, \text{if } y_{c,n} > \gamma | c \in [' stand-by', ' compressor', ' departure', ' cruise', ' arrival']\} \tag{7.7}$$

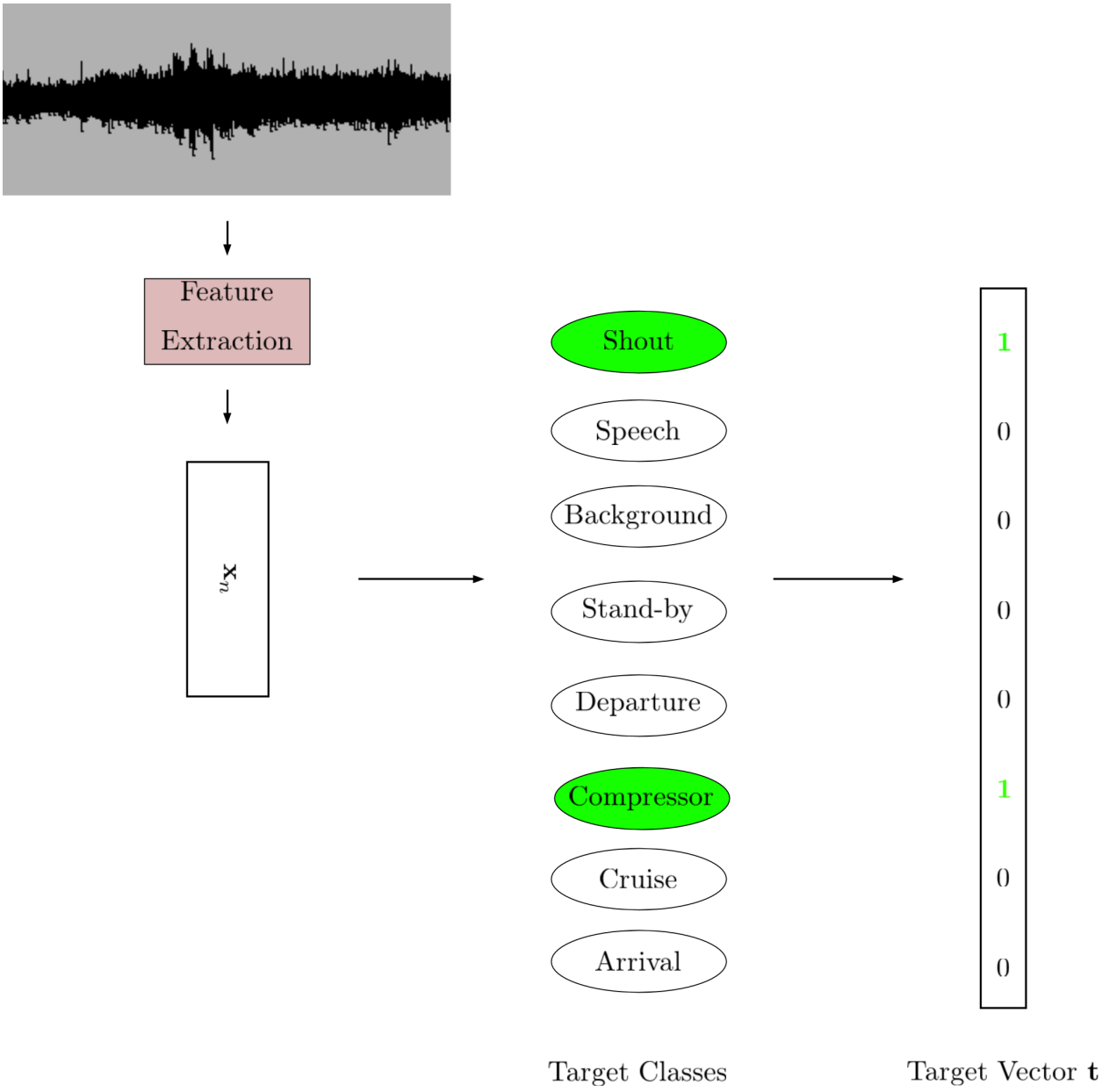

Figure 7.2: Example of a multi-label target vector describing an audio input containing sounds from classes 1 and 5

Here, the classifier is built in such a way that it can handle multiple target classes being on for a single input at step n, since we are in a multi-label framework the target vector now contains more than one non-zeros element, as seen in Figure 7.2. Instead, we can train the network directly on the target classes. The hidden representations describing the classes are now potentially learnt at the same time, with several weight updates coming from two or more error signals for the same input n.

7.3.1 Constraining the output

We can say that our context is very specific in that it does not entirely satisfy the requirement of a true multi-label framework, since all classes can not be on at the same time. We have two distinct sets of classes, speech-related classes and environment-related ones, and at each instant n only one class from each of these two sets can be on. Moreover, the target vectors will therefore always have two non-zero elements, and all classes within one of these two sets are mutually exclusive. These characteristics don't change anything for the training phase, but in the detection stage we add a constraint to our network, in order to force it to output two predictions for each input vector. To do this we force the model to choose exactly one class from each set:

$$\hat{c}_n = \underset{c \in ['background','speech','shout']}{\arg\max} (\mathbf{y}_n) \tag{7.8}$$

$$\hat{e}_n = \underset{c \in ['stan.','comp.','dep.','cru.','arr.',]}{\arg\max} (\mathbf{y}_n) \tag{7.9}$$

Figures 7.3 and 7.4 show the effect of constraining the predictions for two different cases. In Figure 7.3 the original ouput probabilities for voice-related classes were below the threshold, while in Figure 7.4 two of those were above the threshold.

7.4 Experiment Procedure

7.4.1 Data and Features

The data used in these new experiments is the same as used in Chapter 6. As for most experiments in Chapter 5 and Chapter 6, the input vectors are vectors of 40-MFCC coefficients, calculated every $T_i = 10$ ms, over a $T_w = 25$ ms time window. As in previous chapters we split the database in subsets

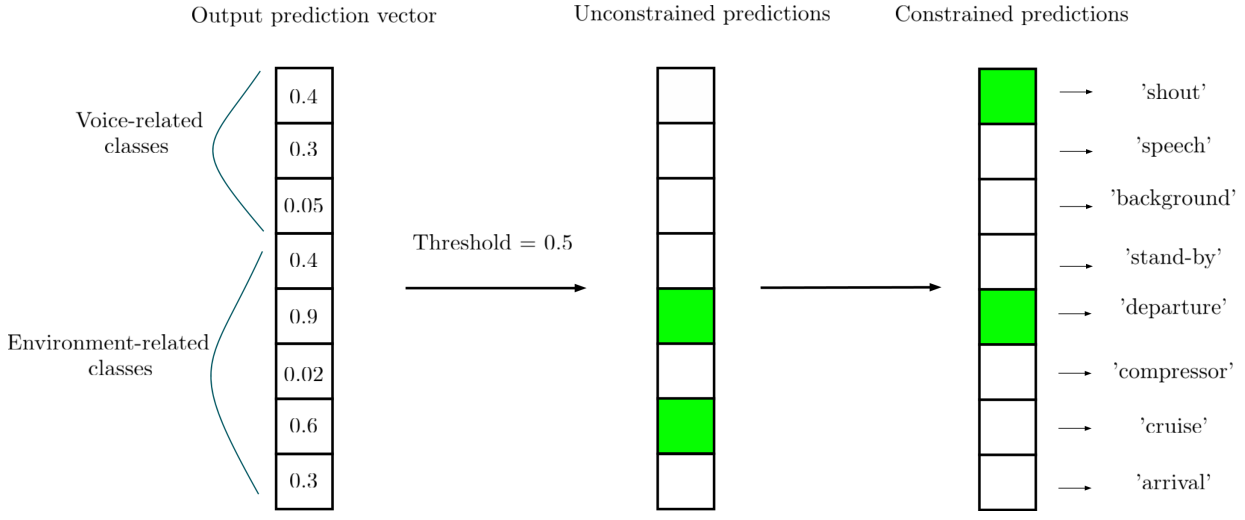

Figure 7.3: Multi-label predictions in the unconstrained and constrained cases. Example with all predictions for voice classes below the threshold

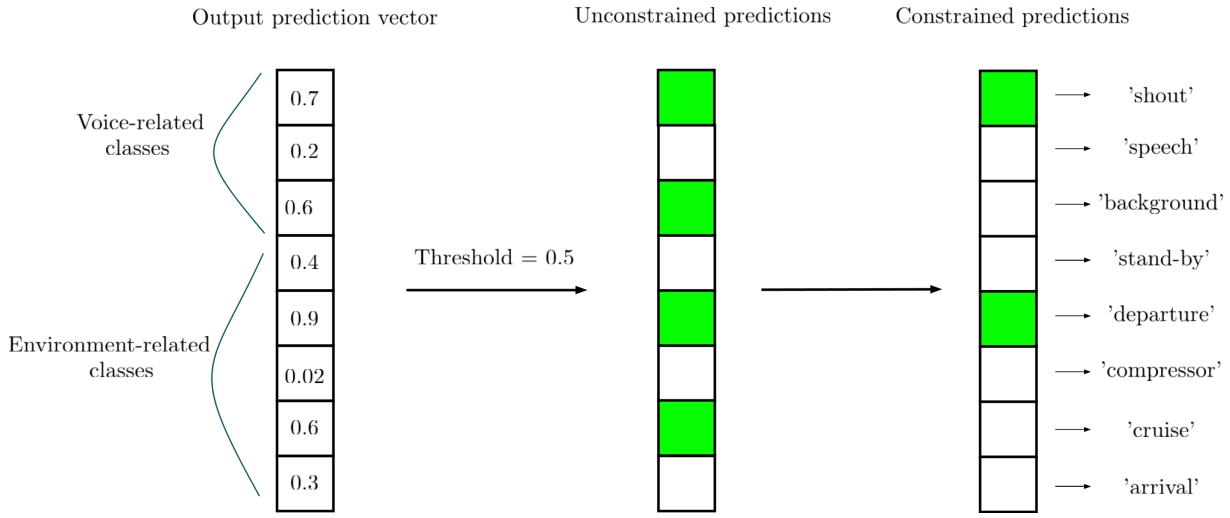

Figure 7.4: Multi-label predictions in the unconstrained and constrained cases. Example with all predictions for voice classes above the threshold

of training, validation and testing data. The distribution of data in seconds for these subsets is the same as in Table 6.1. In this 8-class task, the imbalance problem is also present with a similar impact as in the 3-class task but with lesser consequences than in the 15-class task because the multi-label context does not imply any specific segmentation presenting the risk of having no data for a certain class (as was the case with class 'shout+compressor' in the 15-class task).

7.4.2 ANN settings

In line with the experiments' results from Chapter 6 the ANN configuration used to model the hidden layers is the RNN. Two architectures were tested again, one composed of 1 hidden layer of 128 LSTM units and the other one composed of 1 hidden layer of 256 LSTM units. We first performed estimation with Equation (7.6) for voice classes and Equation (7.7) for environment classes, i.e. in the general case of the multi-label context. Then, we deemed the "restrictive" multi-label more adequate to our problem with equations Equation (7.8) and Equation (7.9) for voice-related classes and environment-related classes respectively.

7.4.3 Training/testing settings

Weight optimization was performed via SGD using the likelihood function defined in Equation (7.5) with the same learning parameters as used in Section 5. A momentum of 0.5 was applied. As described in the precedent chapter, the different configurations introduced in Section 2.1.2 were tested: One with pre-segmented data for the training phase and the test phase (configuration i) and an other one with the pre-segmented data for the training phase and with the streaming data for the test phase. Finally no post-processing was used.

All results for this section will thus be provided through F-score values, as was done in other multi-label studies, [27, 90]. In the single-label case (Chapter 5 and 6) the previous results have been given in the form of a Confusion Matrix where the elements of the diagonal are given by the

accuracy R as defined in Section 2.1.9 and all other elements are false classification (class error). This matrix is not adapted to the multi-label context because it is only able to consider permutation errors, and not insertion/suppression errors as presented in Figure 7.1. Instead of the confusion Matrix, we associate two supplementary metrics to the F-score for each class: the first one is the percentage of accurate detection R (Section 2.1.9), the second one is the ratio of false detection number over the total number of possible false detections for the considered class, in percent (FD). In order to compare the multi-label results with precedent results, we transposed the confusion matrix of Chapter 5 (with equivalent experiment parameters) in term of F-score, Accuracy R and False Detection rate(FD).

7.5 Results

First of all, we give the results for the 3-class problem (Table 5.18 for 1×128 LSTM units and 1×256 LSTM units from Chapter 5) expressed in term of F-score, Accuracy R and False Detection rate FD in configuration i in Table 7.1. Recall that the maximum value of F-score corresponding to a perfect result is 1. Looking at Table 7.1, the conclusions are the same as in Chapter 5: for class shout, the F-score value (79.0%) is inferior to the F-score value of class speech (89.1%), meaning the detection is more efficient for class 'speech'. However the rate of false detections is greater for class 'speech' (10.7%) than for class 'shout' (8.3%). For class 'background', the efficacy is similar (78.0%) to that of class 'shout', as expressed through their R and FD values: Although the $R = 83.6\%$ of class 'background' is greater than $R = 73.6\%$ of class 'shout', the $FD = 24.6\%$ of 'background' is also greater than the $FD = 8.3\%$ of 'shout'. For the 1×256 architecture, as seen in Chapter 5, the results are better if we consider class 'shout': The F-score of 82.1% implies a better detection with a decrease of false detection, the results for classes 'speech' and 'background' remaining approximatively the same.

The first results of Multi-label Classification are given in Table 7.2 which displays the results of a Multi-label RNN with 1×128 LSTM units and 1×256 LSTM units in Configuration i. If we consider the 1×128 LSTM architecture, the F-score values are weak: 62.0% for class 'shout', 66.1% for class 'speech' and 64.3% for class 'background'. For the first two, this is explained by very weak values of accuracy R: 53.8% for 'shout' and 53.2% for 'speech'; the values of FD are respectively weak: 1.5% for class 'shout' and 13.2% for class 'speech'. For class 'background', the accuracy is not bad with $R = 83.6\%$', but at the same time the False Detection rate is $FD = 36.4\%$, implying a F-score equivalent to that of classes 'shout' and 'speech'. If we consider the environment-related classes, their F-scores are slightly higher (except for class 'Arrival'): 86.0% for 'Stand-by', 67.1% for 'Departure', 70.2% for 'Compressor', 86.0% for 'Cruise', and 60.3% for 'Arrival'. 'Stand-by'

Table 7.1: F-score, Accuracy R and False Detection rate FD for the 3-class 'single-'label experiment run with a RNN of 1×128 LSTM units and a RNN of $\times$ 256 LSTM units (configuration i, no post-processing)

	1×128			1×256		
	F-score	R	FD	F-score	R	FD
1: Shout	79.0	73.5	8.3	82.1	78.1	0.5
2: Speech	89.1	86.7	10.7	87.0	82.0	14.5
3: Background	78.0	82.5	24.6	76.2	83.1	15.6

and 'Cruise' have the best results, knowing they are the most represented classes (deduced from Table 6.1) they have a good accuracy R (respectively 79.1% and 90.0%) with few False Detection rates (respectively 3.3% and 11.8%). For the other classes, 'Departure', 'Compressor' and 'Arrival', the weak F-score is due to the weak accuracy value, with R equal to 54.6%, 57.2% and 53.4% respectively, and weak values of FD also: 1.1%, 0.1% and 1.9%.

Looking at the 1×256 architecture, the difference with the 128 architecture is striking, albeit mild; the RNN with 256 units scored similarly or better for all classes, with increases in F-scores from 0% to $+10\%$, increases in R between 0 and 15.4% (to the exception of class 'stand-by' with a loss of 1.4%) and decrease in False Detection rate from 0 to 2.9% (with the exception of class 'Stand-by' again, with a loss of 0.4%). The voice-related class with the highest increase is class 'shout' with an increase in F-score between 62.0% and 71.1%, an increase in R from 5.8% to 62.2% and a decrease of FD from 1.5% to 1.1%. Classes 'speech' and 'background' have a slight increase on these 3 metrics. For environment-related classes, 'Departure', 'Compressor' and 'Arrival' have a significant increase in F-score from 67.1% to 76.4%, from 70.2% to 80.0% and from 60.3% to 69.0% respectively. This progression is mainly due to the improvement in accuracy R: from 54.6% to 69.2%, 57.2% to 72.0% and 53.4% to 65.1%. The variations in FD are very mild with respect to the increase in R: $+0.4\%$ for 'Departure', 0.0% for 'Compressor' and -0.7% for 'arrival'. Modelling power brought by the 256 units allowed the network to handle the complexity of the task much better.

From a general standpoint, considering the score of the voice-related classes, certain results can be explained by the definition our classes (Chapter 5)): the highest FD is for class 'background', which happens to describe sound events that are present in all classes by definition. Knowing that in this multi-label context several classes can be detected at the same time step, we can consider that class 'background' is detected at the same time as classes 'speech' or 'shout'. For class 'speech', the FD can be explained the same way since speech events can be present in class 'shout' and can be detected at the same time.

With respect to the 3 class problem (Table 7.1) the results are less efficient here for the two architectures. If we consider the best case (1×256 LSTM) the F-score decreases from 82.1% to 71.1% for 'shout', from 87.0% to 67.0% for 'speech' and from 76.2% to 68.2% for 'background'. For environment-related classes the FD values are not too high except for the class 'cruise': 11% in the

Table 7.2: F-score, Accuracy R and False Detection rate FD for the 8-class Multi-label experiment run with a RNN of 1×128 LSTM units and a RNN of $\times$ 256 LSTM units (configuration i, no post-processing).

	1×128			1×256		
	F-score	R	FD	F-score	R	FD
1: Shout	62.0	53.8	1.5	71.1	62.2	1.1
2: Speech	66.1	53.2	13.2	67.0	53.2	10.3
3: Background	64.3	83.6	36.4	68.2	88.8	34.3
4: Stand-by	86.0	79.1	3.3	86.3	77.7	2.2
5: Departure	67.1	54.6	1.1	76.4	69.2	1.5
6: Comp.	70.2	57.2	0.1	80.0	72.0	0.1
7: Cruise	86.0	90.0	11.8	87.1	91.9	11.4
8: Arrival	60.3	53.4	1.9	69.0	65.1	1.7

Table 7.3: F-score, Accuracy R and False Detection rate FD for the 8-class Constrained Multi-label experiment run with a RNN of 1×128 LSTM units and a RNN of $\times$ 256 LSTM units (configuration i, no post-processing).

	1×128			1×256		
	F-score	R	FD	F-score	R	FD
1: Shout	65.1	59.2	1.7	71.0	66.2	1.5
2: Speech	69.3	56.9	14.2	71.1	58.1	10.4
3: Background	64.1	86.6	38.7	67.4	90.4	37.5
4: Stand-by	87.2	82.8	5.7	88.1	81.5	3.4
5: Departure	71.0	64.4	2.1	78.0	75.1	2.2
6: Comp.	72.2	60.6	0.1	79.0	72.0	0.2
7: Cruise	86.4	94.6	14.6	87.2	95.6	14.2
8: Arrival	63.0	61.3	2.5	71.1	70.1	2.2

128 units case and 11.4% in the 256 units case. Regarding the 256 units architecture, the results are good or acceptable for the 3 metrics (expected for class 'arrival' with R equal to 35.1%)

The results of the constrained Multi-label are presented in Table 7.3. Considering the 1×128 LSTM units, enforcing the constraint presented in Section 7.3.1 improved all F-scores by between 0 and $+3\%$. Although the accuracy R increase between 3% and 10%, the false detections FD were all slightly increased by 0 to 3% which slowed down the F-score progression. For the class 'shout', we obtained a F-score of 65.1%, weak accuracy $R = 59.2\%$ and a False Detection rate FD of 1.70% slightly increasing the scores from the unconstrained case with a F-score = 62.1%, a R = 53.8% and a FD = 1.50%. With a 1×256 LSTM units architecture for the constrained Multi-label, the constraint had the same effect, raising accuracy for all classes by 0 to 6% and F-scores by $+1\%$to $+7\%$. Again, False Detection rate were higher by 0 to 3% for all classes. Concerning class 'shout' we obtained a F-score of 71.0%, an accuracy R = 66.2% and a False detection rate $FD = 1.50\%$, performing a little better than the unconstrained case with 256 units: F-score= 71.1%, R = 62.2% and FD = 1.10%.

As a conclusion, choosing the best two predictions from the two subsets of classes (voice-related and environment-related) increased both the accuracy and the false alarm rate, which has a very interesting explanation. The constraint has two distinct effects on the result, as exemplified in Figures 7.3 and 7.4. Consider one of the two subsets on which the constraint is enforced. If in the unconstrained case the predictions were all below the threshold, as in Figure 7.3, the constraint either increases accuracy or increases false alarms. If in the unconstrained case the predictions were all above the threshold, as in Figure 7.4 the constraint then either decreases accuracy or decreases false alarms. Here, since both accuracy and false alarms increases, our unconstrained predictions were below the threshold in the cases making the difference between the unconstrained and constrained case.

Finally, it seems that the constrained multi-label RNNs scored well on classes with a high representation such as 'background', 'stand-by' and 'cruise' with accuracy above 80% for all three with both 128 and 256 units. However the false alarm rates for these classes was quite high (38.7% and 37.5% for 'background' and 5.7% and 3.4% for 'stand-by', 14.6% and 14.2% for 'cruise') as compared to other classes (2.2% was the maximum false alarm rate, for class 'departure'). This speaks to the model's behavior, which relies on its multi-label capability to output the bigger classes as

often as possible, inducing false alarms, in order to get more chances of accuracy predicting them, while not impacting accuracy for other classes since it can also output other classes. Class 'speech' was a counter-example of all this though, as it had low detection accuracy despite being the largest voice-related class. Strangely, its false alarm rate was quite high (14.2% and 10.4%) as though it was being treated the same way as other big classes were, as we just explained. This remains a gray area for us to explore.

Neither architecture, even with constrained output, performed on par with the 3-class RNNs from Chapter 5 in terms of F-score and accuracy (-15% to -30%), although the 256-units multi-label RNN managed to outperform both of them by 5% in terms of accuracy for class 'background'. This came at the price of higher false alarm rates though ($+23\%$), showing the difficulty of multi-label networks to balance between accuracy and false alarm. Constraining the predictions was not enough to catch up with the single-label 3-class RNNs. Finally, the best detection accuracy for class 'shout' is still provided by a single-label 256-hidden units RNN on the 3-class task, as in Chapter 5, with accuracy of 78.1% and a very low false alarm rate of 0.5%.

7.6 Conclusion and Discussion

In this chapter we use the multi-label paradigm to classify speech-related classes and environment-related classes in parallel. This paradigm is suited to our task since it allows the model to output multiple classes at the same time, and our labeling scheme includes two sets of (mutually exclusive) classes describing sounds occurring simultaneously. The multi-label therefore allows the model to detect classes from these two sets of classes at the same time, without requiring any fusion between them as was the case in Chapter 6. We outlined the mathematical framework on which this paradigm relies, and explained the consequences of multiple output predictions on the error count and the performance measurement. We described the target classes used in our experiments and explained how these classes were fitting in the multi-label context. All speech-related classes are mutually exclusive, as well as all environment-related ones, but each input is associated with one class from each subsets. Therefore our models were trained to output one class from each subsets at every time-step. In the first experiment we did not constrain the model, allowing it to output as many classes as it wished. In the second experiment we forced to output one class from each subset by choosing the one with the highest estimated posterior probability.

Judging from the results from Section 7.2 it seems that the complexity of the task compared to direct classification (3-class task) is too great, and the burden of adding 5 more classes to be detected at the same time overweight the added-value of the contextual information brought by these 5 classes. As a result the network has now to perform a harder task with the same resources as before, which explains why it failed in detecting the target classes. The issue of the threshold value, pointed out by the constrained results in Section 7.5, leads us to think that an evaluation should be carried out on how to better select this value and how it impacts accuracy and false alarms. The trade-off between accuracy and false alarms for our multi-label models was too great, but a deeper investigation might help reduce it. More specifically the constraint introduced in Section 7.3.1 could be learnt in the training phase by using two separate sets of softmax output units (instead of the sigmoid ones). One set would be used to normalize the output units corresponding to the speech-related classes and the other one to normalize the output units corresponding to the environment-related classes. This would ensure that exactly one speech class and one environment class are predicted at each time step. The mathematical relations carried in the softmax are given by Equation. (7.10) and (7.11):

$$y_c = \frac{\exp -(\mathbf{w}_c \mathbf{h}^{(L)} + w_{c0})}{\sum_{i=1}^{3} \exp -(\mathbf{w}_i \mathbf{h}^{(L)} + w_{i0})}, \text{ for } c \in [1 :' \mathit{shout}', 2 :' \mathit{speech}', 3 :' \mathit{background}'] \tag{7.10}$$

for voice-related classes and the equivalent for environment-related classes:

$$y_c = \frac{\exp -(\mathbf{w}_c \mathbf{h}^{(L)} + w_{c0})}{\sum_{i=1}^{C} \exp -(\mathbf{w}_i \mathbf{h}^{(L)} + w_{i0})}, \text{ for } c \in [1 :' \mathit{std-by}', 2 :' \mathit{comp.}', 3 :' \mathit{depart.}', 4 :' \mathit{crse}', 5 :' \mathit{arr.}',] \tag{7.11}$$

These new output relations imply a modification of the likelihood definition used in the learning phase.

We did not consider the environment-related classes to be mutually exclusives here because their successive realizations are not arbitrary and follow a well structured pattern. Indeed each of the environment event defined in Section6.1 occurs following a temporal pattern linked to the functioning of the transportation means (i.e in our case the metro). This pattern is identified in Figure 4.5.

Chapter 8

Conclusion

This study on the Detection of screams and shouts in public transportation was conducted in the framework of audio-visual automatic surveillance. As seen in the introduction, video surveillance, which to this day remains the most exploited method, has limitations in embedded environments such as changes in lighting conditions, obstruction of the line of sight, or physical limitations of the visual field. The analysis of audio signal is then used to overcome these limitations. More specifically, our study aims at automatically detecting violent situations based on the audio stream captured from a public transportation environment, based on the recognition of screams and shouts.

In Chapter 2 we described the classification paradigm adopted, in which the system tries to identify a pattern within the stream. This classification is based on the analysis and extraction of the signal's characteristics. When the considered signal is made of a stream of successive data, the analysis process implies an arbitrary segmentation of the said signal. If the target patterns evolve through time (i.e sequential patterns), the succession of the characteristics extracted needs to be analyzed to recognize them. We explained that audio patterns are such sequences, defined by the evolution of their spectral properties through time, and that their duration is not fixed which brings ambiguity to the classification process. Since the target pattern are known the task is said to be supervised, and a learning phase is required to train a statistical model to either learn how to characterize some patterns within the data (generative case) or to differentiate them (discriminative case). Therefore, some generative models were introduced (GMM and HMM) as well as some discriminative ones (SVM and ANN). After having outlined the post-processing techniques used later in the study, as well as the metrics used to evaluate our systems, we presented many previous studies tackling various Classification/Detection problems, emphasizing the diversity of applications and the prominence of this topic in the field of audio signal processing. Indeed, numerous works touching upon the automatic detection of audio events have been applied to robotics, speech recognition, acoustic scene analysis and audio surveillance. Finally we mentioned that in a realistic environment many interfering sources make up the acoustic environment, depending on the targeted application.

Having decided to use Artificial Neural Networks to pursue our task, we outlined their underlying mathematical framework in Chapter 3. The logistic regression model, which constitutes the basis of ANNs, was introduced first, leading to the basic unit composing them; the neuron. We showed how multiple of these units can be put together in a graphical network of one or several layers and connected to each other via weights, to form a NN. Each unit analyzes the information from other units located in the previous layer, or from the input data for units located in the first layer, by means of a mathematical transformation outputting binary values (0 or 1). If more than one layer is used the NN is called a Deep Neural Network (Deep NN). We gave the general principles of the training process according to which the weights are learned via maximization of a likelihood function. The latter is achieved by means of an efficient algorithm called Back-Propagation, propagating the prediction errors made by the classifier backwards into all layers so that neurons in those layers can adapt their weights accordingly. We showed that the NN architecture can be extended to analyze the temporal dimension of the signal through the use of filters, in another architecture called Convolutional Neural Networks (CNN). The input is formatted in a 2-dimensional matrix and the filters, composed of weights similar to the ones used in a NN, are trained to identify patterns across both dimensions. We showed that this architecture allows to display both the signal's spectral properties on one axis and temporal on the other one, and to learn spectro-temporal patterns. Finally we introduced another architecture, adding a temporal decoding scheme to the NN, implemented via recurrent connections between neurons. These

connections are established through another set of weights which are trained to identify temporal properties of patterns in the input signal.

All the models introduced in Chapter 3 are supervised and require a training stage to learn from a training database. In Chapter 4 we gave the details about the creation of our training database, which we also used to test our models. Within the framework of research project Dé-GIV, we captured the acoustic environment from a metro in service, where a realistic environment was re-created by actors to provide a simulation of violent situations in different scenarios. Each scenario was described, outlining the variety of contexts used (purse snatching, robbery, assault, quarrel) and the variability of sound occurrences through the number of actors involved. We manually labelled the hour of resulting audio, assigning labels to the entirety of the audio signal, according to the sonic content perceived by listening to it. From this labelling task it came out that the situations to be detected in our application can be characterized by two types of events, which we identified as screams and shouts. The former is a high energy vocal signal without intelligible speech content, with a rather flat temporal signature. The latter describes quite energetic vocal signals with intelligible speech content, and are related to quarrels or altercation between people. We provided an analysis of this database, showing the diversity of the acoustic background environment and outlining some characteristic sound events composing it (door signal, brakes noise, etc..). The resulting mix of these sources displayed a high variability, as well as distinct temporal patterns, which adds to the complexity of the task.

Chapter 5 In Chapter 5 we used three classes describing the voice content of the signal to classify the audio signal; a 'shout' class which included all ocurrences of voiced signal related to altercations (namely screaming and shouting), another class labelled 'speech' for all signals with voice content not corresponding to the 'shout' class, and one last class labelled 'background' to describe all other sounds (i.e without vocal content). These classes were associated with the database, in the form of a label assigned to each audio sequence, and used to train the Neural Network models described in Chapter 3. The first model consisted of a NN, which analyzes the current input as a whole. The model was first used to test two different feature representations of the signal in order to choose the one which yielded the best results. Based on the chosen feature representation we then varied the NN's architecture, adding layers of neurons and modifying the number of neurons in each, to bring more modelling power to the network and see to what extent the complexity could be extended, given our database. Context windows were used to take into account the temporal context, concatenating successive input frames together so the classifier can assess the signal's temporal evolution. With the architecture selected earlier, different sizes of that window were tested to assess the impact of temporal information. In this section the frames were taken from pre-segmented sequences within our dataset, meaning they all belonged to one same sequence. This way of processing the dataset was referred to as Configuration i. We tried improving the optimization of our model's weights by generatively pre-training them on unlabelled data as explained in Chapter 3, before discriminatively fine-tuning them as before. Lastly, we tested the two post-processing methods presented in Chapter 2, improving the model's predictions. We then used the Convolutional Neural Network model, where inputs are considered as 2-dimensional, including a time axis, and analyzed across both dimension by 2-D filters. These filters, taking into account the temporal dimension, were trained to detect temporal patterns and spectral patterns, using a fixed architecture (number of layers and neurons). Again different lengths of context window were used to assess the impact of temporal information. With the selected context window length we carried out experiments to assess the impact of the filters dimension on the recognition of

our target patterns. The last neural model considered in Chapter 3, the RNN/LSTM, implemented a temporal decoding scheme by learning the succession of events explicitely through recurrent connections between its neurons, as explained in Chapter 3. The LSTM units added precision to this by allowing temporal information to be stored in a separate memory pipe, controlling the way information is extracted from the input signal by way of filtering it through three gates, each one having a distinct purpose (select information to use from the input and the memory pipe, select information from that input to store in the memory pipe, and select what to output as prediction). From the most basic architecture (1 hidden layer with 128 units) we tried increasing the modelling power by adding neurons in that layer, and adding a second layer on top. Finally, to implement a more realistic setting, we identified another way of formatting the data when concatenating frames together in the context window, where the segmentation is no longer performed on sequences but rather on the incoming stream regardless of the classes which each frame belongs to. This approach, referred to as streaming, implies that inputs can consist of frames from different patterns.

The results obtained with the NN show that with 1 layer of 128 neurons it is possible to reach a high detection rate on a pre-segmented database for class 'shout' (more than 80%) while attaining high rates for the remaining classes (over 80% for class 'speech' and over 70% for class 'background'), at the expense of a long context window. The NN appears to be able to characterize the audio environment of embedded transportation to a good degree of accuracy, in terms of voiced signal against other sounds. The downside of this experiment is the length of the contextual window ($500ms$) required to achieve these results which can be prohibitive in some contexts. Moving to the CNN, using 15×15 filters, the total error rate decreased, accuracy improved for class 'shout' ($+3\%$) and class 'background' ($+6\%$), although it slightly decreased for class 'speech' (-2%), indicating that a distinct analysis of the time and frequency dimensions increased the modelling power of the classifier. The improvement was small yet sizeable, enough to clearly show the benefits gained from the 2-dimensional analysis. Nonetheless, the use of filters on fixed-sized inputs still does not account for the varying size of the patterns. Max-pooling the filters' output was a way of overcoming these variations in size, but only to a certain extent. Our best RNN model, with 1 layer of 256 units, lowered the total error rate even further, although also impairing detection accuracy for class 'shout' with a 7% decrease compared with the previous model. Nonetheless its modelling power seemed vastly superior to the previous models, as shown by the performance on the training set once learning was complete, where accuracy for all classes was greater than 90%. This indicates that if more care is taken to improve generalization, the performance might increase. Given our results so far the RNN seemed to have the higher potential. The three models presented thus far had rather low false alarm rates for class 'shout', with less than 2% for the NN and RNN and less than 4% for the CNN, which is a pivotal point to be considered in real-world applications. For medical purposes for instance, false alarms are virtually prohibited as their consequence can be disastrous. In our application however the implication of false alarms can be regarded less harmful, but this depends on engineering considerations. All tests carried out in realistic configurations (Configurations ii and iii) have identified the limitations of our best model (the RNN), who looses its discriminative power when the signal is no longer pre-segmented, although the accuracy for the other two classes remains very good however.

Chapter 6 In Chapter 6 to achieve the same task we looked at the problem under a different light and decided to analyze the sonic environment, in order to bring some information about it to the classifier, hoping it would help it in its original task. We identified the main sound events composing the environment to create 15 sub-classes of the voice-related classes according to those

events, ending up with more accurately defined classes than the original voice-related classes. To carry out experiments with this new labelling scheme we used the best model from Chapter 5, the RNN/LSTM, with two different architectures (128 and 256 units) to see if we could gain from more complexity. We classified the 15 subclasses directly with the network, using pre-segmented data to evaluate the performance first, then testing the same trained network on a stream of data without pre-segmentation. The latter gave us an insight on the model's ability to deal with real data. To see if the use of smaller subclasses helped the classifier in classifying the data into the three original speech-related classes ('shout', 'speech', 'background') we marginalized each result over the environment-related classes to compute the classification accuracy. In order to see if this impacted the classification of the original environment-related classes ('stand-by', 'departure', 'compressor', 'cruise', 'arrival') we also marginalized the results over the speech-related classes.

Results showed that such partitioning of the voice-related classes adds precision in the class definition, resulting in higher classification accuracy in pre-segmented mode (configuration i), confirming the importance of the environmental noise inherent to our application. Unfortunately, under realistic conditions, analyzing the incoming audio stream without pre-segmentation (Configurations ii and iii) the effect of partitioning the voice-related classes was detrimental to the detection of 'shout' with significant decreases in accuracy from 68.6% to 52.4% , as well as class 'speech' (76.5% vs 85.7%). The 15 compound classes resulting from the creation of sub-classes turned out to reduce the size of the individual sequences, which makes the task harder in those realistic configurations.

Chapter 7 To benefit from the contextual information brought by the environment labels without suffering from the 15-class complexity or the small size of sequences, we adopted a different strategy in Chapter 7. The classifier was given the possibility of recognizing multiple events at a time, in the hope that it would be able to classify the voice-related classes along with the environment-related ones. This paradigm is called multi-label, and induces a slightly different mathematical framework, which we described in the chapter. Contrary to what was done so far the training data carried several target labels per input. We used the same model as before, with the same architecture; a RNN with 128 or 256 units. In the first experiment the network was given full liberty to output as many classes as it wished. In a second experiment we constrained its output so as to have one prediction for speech-related classes and another for environment-related classes. This idea is based on the fact that our labelling framework is not a common multi-label framework, where all classes are independent one another and can occur independently at any time. Here, all speech-related classes are mutually exclusive, and so are the environment-related ones. Therefore all inputs are defined by exactly two labels, which is why it appeared to us that the model should be given some indication as to this particularity. The constraint was only enforced during testing and not during training.

Initial experiments showed that the degree of freedom given by the inherent ability of the network to output several classes was hard to control, and was influenced by the imbalance of our dataset. Indeed, even with a constrained output, the model had a tendency to output frantically classes with high representation (i.e 'background', 'stand-by' and 'cruise' with respectively $90.4\%, 81.5\%, 95.6\%$ accuracy with the 256-units RNN), although this was not verified for class 'speech' (with only 58.1% accuracy), at the cost of high false alarm rates. It appears to us that the mathematical framework of the training procedure does not provide enough incentive for the model to learn to accurately detect classes without creating false alarms, since the error stemming from a false alarm is compensated for when the right class is also predicted. In other words,

falsely predicting a class does not necessarily imply that the correct class was not predicted as well. Conversely, the model struggled to learn from classes with lower representation since potential false alarms created error signals which were then used to adjust the weights. In other terms, the multi-label classifier has the inherent ability to be right and wrong at the same time, rendering the learning process via gradient descent harder.

8.0.1 Perspective

Expanding the database. The most obvious direction in which to look next, based on the results of this study, is the expansion of the database. As we have exposed, the lack of training data did not allow us to benefit from deep architectures. More specifically when creating the compound classes the scarcity of certain events did not permit us to have enough data to train our models with. Moreover, this approach led to shorter events, rendering learning of the temporal structures more difficult, especially in Configurations ii and iii where no pre-segmentation was involved.

Leveraging big data: Transfer Learning In the course of our experiments with generative pre-training, we contemplated the idea of training RBMs with unlabeled data from different databases. Having access to data from a different environment containing similar sound sources could give the model a different insight on the sources, helping it find different representations of those sources, which it would not have found with the original database yet might be helpful in its classification task. This would be similar to what is achieved by Transfer Learning [28, 82, 89] where the knowledge from a task is used to complete a different task. For ANNs, transfer learning amounts to using a network already trained on a specific kind of data to either re-train it or even just use it to perform a classification/regression task on different data. Similarly, data augmentation can be used to benefit from having more training data [74], by creating synthetic data.

Dealing with false alarms. As low as the false alarm rate can be it still represents a hurdle to the implementation of such classifiers in a real-world application. It would be necessary to apply some sort of penalty for false alarms during the learning phase in order to guide the classifier in a way which would force him to avoid making such false alarms. This penalty could be embodied by a higher cost value, leading to a bigger step in the direction of the gradient during SGD.

Honing on the RNN training settings. The incredible learning capabilities displayed by the RNN, adapting to the target classes very well during training with less than 4% of errors for each class on the training data, indicate that a further investigation of the training procedure of the RNN might yield even better results. Elements such as learning rates, momentum, as well as the number of epoch, could have a strong impact, and although the training time for RNN was prohibitive here it would be interesting to delve more into these settings in the future.

Constraining the ouput in multi-label training. As mentionned earlier the multi-label paradigm deserves more investigation, and more specifically in our particular setting where all classes in our two sets of classes (speech-related and enviroment-related) are mutually exclusive. Moreover, since these two sets of classes consitute a full basis of our audio scene (i.e all frames of audio signal belong to one and only class in each of the two subsets) exactly one class from each set should be predicted by the network. As we saw, the independent sigmoid units leads to too many false alarms or too little accuracy. This might be caused by the fact that the model does not learn the

targets 'jointly', as opposed to a single-label paradigm. As explained in 7.2 all false alarms need to be accounted for in the optimization process, on top of the missed detection, which may lead to more complexity. This is why we believe that adding a constraint at the ouptut layer, in the form of two sets of softmax units, one for each set of classes (speech-related and enviroment-related) might alleviate the burden by reducing the number of potential errors. The implementation of this idea, carried out in Section 7.3.1 by enforcing a constraint during classification of the test set, did not include it in the training stage. The next step would be to enforce this constraint in the learning process to help the model learn the two sets of classes in parallel.

Analyzing the succession of classes. The real-case approach adopted in Configurations ii and iii, introducing the concept of a continuous stream of audio without segmentation, outlined the weakness of our model when confronted with such situations. The difference in performance with Configuration i, which implied previous segmentation of the audio sequences, indicated that the models are capable of identifying the internal structures of classes, but struggle when they have to analyze the succession of events from different classes. It would be interesting to combine two models where one would be dedicated to learning the successions of classes, in order to help the other one to find the boundaries between classes within the audio stream.

Bibliography

[1] O. Abdel-Hamid, A.R Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, *Convolutional neural networks for speech recognition*, IEEE/ACM Transactions on Audio, Speech, and Language Processing **22** (2014), no. 10, 1533–1545.

[2] J. Baker, *The dragon system–an overview*, IEEE Transactions on Acoustics, Speech, and Signal Processing **23** (1975), no. 1, 24–29.

[3] D. Barchiesi, D. Giannoulis, D. Stowell, and M. D. Plumbley, *Acoustic scene classification: Classifying environments from the sounds they produce*, IEEE Signal Processing Magazine **32** (2015), no. 3, 16–34.

[4] F. Bastien, P. Lamblin, Pascanu R., I.J Goodfellow J. Bergstra, Bergeron A., N. Bouchard, and Bengio Y., *Theano: new features and speed improvements*, Deep Learning and Unsupervised Feature Learning NIPS Workshop, 2012.

[5] Y. Bengio, *Learning deep architectures for ai*, Found. Trends Mach. Learn. **2** (2009), no. 1, 1–127.

[6] Y. Bengio, P. Frasconi, and P. Simard, *The problem of learning long-term dependencies in recurrent networks*, IEEE International Conference on Neural Networks, 1993, pp. 1183–1188 vol.3.

[7] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, *Greedy layer-wise training of deep networks*, IN NIPS (Vancouver, B.C., Canada), MIT Press, 2007.

[8] Y. Bengio, P. Simard, and P. Frasconi, *Learning long-term dependencies with gradient descent is difficult*, IEEE Transactions on Neural Networks **5** (1994), no. 2, 157–166.

[9] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, *Theano: a CPU and GPU math expression compiler*, Proceedings of the Python for Scientific Computing Conference (SciPy), Jun. 2010.

[10] C. M. Bishop, *Neural networks for pattern recognition*, ch. The Multi-layer Perceptron, pp. 116–161, Oxford University Press Inc., New York, USA, 1995.

[11] V. Bisot, R. Serizel, S. Essid, and G. Richard, *Acoustic scene classification with matrix factorization for unsupervised feature learning*, 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), March 2016, pp. 6445–6449.

[12] B.Mathieu, S.Essid, T.Fillon, J.Prado, and G.Richard, *Yaafe, an easy to use and efficient audio feature extraction software*, proceedings of the 11th ISMIR conference (Utrecht, Netherlands), 2010.

[13] Hervé A. Bourlard and Nelson Morgan, *The hybrid hmm/mlp approach*, pp. 155–183, Springer US, Boston, MA, 1994.

[14] Brian McFee, Colin Raffel, Dawen Liang, Daniel P.W. Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto, *librosa: Audio and Music Signal Analysis in Python*, Proceedings of the 14th Python in Science Conference (Kathryn Huff and James Bergstra, eds.), 2015, pp. 18 – 25.

[15] S. Chu, S. Narayanan, C. c. J. Kuo, and M. J. Mataric, *Where am i? scene recognition for mobile robots using audio features*, 2006 IEEE International Conference on Multimedia and Expo, July 2006, pp. 885–888.

[16] S. Chu, S. Narayanan, and C.C. Jay Kuo, *Environmental sound recognition with time-frequency audio features*, IEEE Trans. on Audio, Speech, and Language Processing **17** (2009), no. 6, 1142–1158.

[17] Avery Li chun Wang and Th Floor Block F, *An industrial-strength audio search algorithm*, Proceedings of the 4 th International Conference on Music Information Retrieval, 2003.

[18] C. Clavel, L. Devillers, G. Richard, I. Vasilescu, and T. Ehrette, *Detection and analysis of abnormal situations through fear-type acoustic manifestations*, 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07 (Honolulu, Hawaii, USA), vol. 4, April 2007, pp. IV–21–IV–24.

[19] C. Clavel, T. Ehrette, and G. Richard, *Events detection for an audio-based surveillance system*, 2005 IEEE International Conference on Multimedia and Expo (Amsterdam, The Netherlands), July 2005, pp. 1306–1309.

[20] Michael Cowling and Renate Sitte, *Comparison of techniques for environmental sound recognition*, Pattern Recognition Letters **24** (2003), no. 15, 2895 – 2907.

[21] M. Crocco, M. Cristani, A. Trucco, and V. Murino, *Audio surveillance: A systematic review*, ACM Comput. Surv. **48** (2016), no. 4, 52:1–52:46.

[22] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of Control, Signals and Systems **2** (1989), no. 4, 303–314.

[23] G.E. Dahl, Dong Yu, Li Deng, and A. Acero, *Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition*, Audio, Speech, and Language Processing, IEEE Transactions on **20** (2012), no. 1, 30–42.

[24] A. P. Dempster, N. M. Laird, and D. B. Rubin, *Maximum likelihood from incomplete data via the em algorithm*, JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B **39** (1977), no. 1, 1–38.

[25] L. Deng and D. Yu, *Deep learning: Methods and applications*, Found. Trends Signal Process. **7** (2014), no. issue 3-4, 197–387.

[26] J. Dennis, H.D Tran, and E.S Chang, *Image feature representation of the subband power distribution for robust sound event classification*, IEEE Transactions on Audio, Speech, and Language Processing **21** (2013), no. 2, 367–377.

[27] A. Diment, E. Cakir, T. Heittola, and T. Virtanen, *Automatic recognition of environmental sound events using all-pole group delay features*, European Signal Processing Conference (Nice, France), Aug. 30 - Sept. 4 2015, pp. 734–738.

[28] A. Diment and T. Virtanen, *Transfer learning of weakly labeled audio*, IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (New Paltz, NY), Oct. 2017.

[29] A. J. Eronen, V. T. Peltonen, J. T. Tuomi, A. P. Klapuri, S. Fagerlund, T. Sorsa, G. Lorho, and J. Huopaniemi, *Audio-based context recognition*, IEEE Transactions on Audio, Speech, and Language Processing **14** (2006), no. 1, 321–329.

[30] Slim Essid, Marine Campedel, Gaël Richard, Tomas Piatrik, Rachid Benmokhtar, and Benoit Huet, *Machine learning techniques for multimedia analysis*, pp. 59–80, John Wiley & Sons, Ltd, 2011.

[31] Slim Essid, Sanjeel Parekh, Ngoc Q. K. Duong, Romain Serizel, Alexey Ozerov, Fabio Antonacci, and Augusto Sarti, *Multiview approaches to event detection and scene analysis*, pp. 243–276, Springer International Publishing, Cham, 2018.

[32] M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim, *Do we need hundreds of classifiers to solve real world classification ?*, J. of Machine Learning Research **15** (2014), 3133–3181.

[33] Asja Fischer and Christian Igel, *Progress in pattern recognition, image analysis, computer vision, and applications: 17th ibero american congress, ciarp 2012, buenos aires, argentina, september 3-6, 2012.*, no. LNCS 7441, ch. An Introduction to Restricted Boltzmann Machines, pp. 14–36, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[34] K. Fukushima, *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*, Biological Cybernetics **36** (1980), no. 4, 193–202.

[35] K.i. Funahashi, *Multilayer neural networks and bayes decision theory*, Neural Networks **11** (1998), no. 2, 209 – 213.

[36] F. Ganansia, V. Delcourt, QC. Pham, A. Lapeyronnie, C. Baudry, L. Lucat, P. Sayd, S. Ambellouis, D. Sodoyer, AC. Barcelo, and F. Heer, *Audio-video surveillance system for public transportation*, World Congress on Railway Research (Lille, 2011, France), May 22-26 2011.

[37] J. T. Geiger and K. Helwani, *Improving event detection for audio surveillance using gabor filterbank features*, European Signal Processing Conference (Nice, France), Aug. 30 - Sept. 4 2015, pp. 719–723.

[38] L. Gerosa, G. Valenzise, M. Tagliasacchi, F. Antonacci, and A. Sarti, *Scream and Gunshot detection in noisy environments*, European Signal Processing Conference (Poznan, Poland), Sep. 3-7 2007.

[39] F.A Gers, J. Schmidhuber A., and F.A Cummins, *Learning to forget: Continual prediction with lstm*, Neural Comput. **12** (2000), no. 10, 2451–2471.

[40] F.A Gers, N.N Schraudolph, and J. Schmidhuber, *Learning precise timing with lstm recurrent networks*, J. Mach. Learn. Res. **3** (2003), 115–143.

[41] Zoubin Ghahramani and Michael I. Jordan, *Supervised learning from incomplete data via an em approach*, Advances in Neural Information Processing Systems 6, Morgan Kaufmann, 1994, pp. 120–127.

[42] H. Gish, *A probabilistic approach to the understanding and training of neural network classifiers*, International Conference on Acoustics, Speech, and Signal Processing (Albuquerque, NM, USA), Apr 1990, pp. 1361–1364 vol.3.

[43] A. Graves, A. r. Mohamed, and G. Hinton, *Speech recognition with deep recurrent neural networks*, 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, May 2013, pp. 6645–6649.

[44] A. Graves and J. Schmidhuber, *Framewise phoneme classification with bidirectional lstm and other neural network architectures*, Neural Networks (2005), 5–6.

[45] Alex Graves, *Sequence transduction with recurrent neural networks*, 2012 International Conference of Machine Learning (Edimburgh, Scotland), 2012.

[46] Alex Graves and Juergen Schmidhuber, *Offline handwriting recognition with multidimensional recurrent neural networks*, Advances in Neural Information Processing Systems 21 (D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds.), Curran Associates, Inc., 2009, pp. 545–552.

[47] Stephen Grossberg, *Some networks that can learn, remember, and reproduce any number of complicated space-time patterns, i*, Indiana Univ. Math. J. **19** (1970), 53–91.

[48] H. He and E.A. Garcia, *Learning from imbalanced data*, IEEE Trans. on Knowl. and Data Eng. **21** (2009), no. 9, 1263–1284.

[49] T. Heittola, A. Mesaros, A. Eronen, and T. Virtanen, *Context-dependent sound event detection*, EURASIP Journal on Audio, Speech, and Music Processing **2013** (2013), no. 1, 1.

[50] Perfecto Herrera, G. Peeters, and S. Dubnov, *Automatic classification of musical instrument sounds*, Journal of New Music Research **32** (2003).

[51] G. E. Hinton, S. Osindero, and Y.-W. Teh, *A fast learning algorithm for deep belief nets*, Neural Comput. **18** (2006), no. 7, 1527–1554.

[52] G. E. Hinton and T. J. Sejnowski, *Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1*, ch. Learning and Relearning in Boltzmann Machines, pp. 282–317, MIT Press, Cambridge, MA, USA, 1986.

[53] S. Hochreiter and J. Schmidhuber, *Long short-term memory*, Neural Comput. **9** (1997), no. 8, 1735–1780.

[54] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber, *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*, 2001.

[55] W. Huang, T.-K. Chiew, H. Li, T. S. Kok, and J. Biswas, *Scream detection for home applications*, IEEE Conference on Industrial Electronics and Applications (Taichung, Taiwan), 2010, pp. 2115–2120.

[56] A.G. Ivakhnenko, *The group method of data handling: an rival of the method of stochastic approximation*, Soviet Automatic Control 13(3) (1968), 43–55.

[57] M. Janvier, X. Alameda-Pineda, L. Girin, and R. Horaud, *Sound-event recognition with a companion humanoid*, IEEE-RAS International Conference on Humanoid Robots (Humanoids), Nov 2012, pp. 104–111.

[58] M. Janvier, X. Alameda-Pineda, L. Girin, and R. Horaud, *Sound Representation and Classification Benchmark for Domestic Robots*, 2014 IEEE International Conference on Robotics and Automation (ICRA 2014) (Hong-Kong, China), IEEE, May 2014, pp. 6285–6292.

[59] Nathalie Japkowicz and Shaju Stephen, *The class imbalance problem: A systematic study*, Intell. Data Anal. **6** (2002), no. 5, 429–449.

[60] F. Jelinek, *Continuous speech recognition by statistical methods*, Proceedings of the IEEE **64** (1976), no. 4, 532–556.

[61] K. Kim and H. Ko, *Hierarchical approach for abnormal acoustic event classification in an elevator*, 2011 8th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS) (Klagenfurt, Austria), Aug 2011, pp. 89–94.

[62] T. Komatsu, Y. Senda, and R. Kondo, *Acoustic event detection based on non-negative matrix factorization with mixtures of local dictionaries and activation aggregation*, 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), March 2016, pp. 2259–2263.

[63] G. Lafay, M. Lagrange, M. Rossignol, E. Benetos, and A. Roebel, *A morphological model for simulating acoustic scenes and its application to sound event detection*, IEEE/ACM Transactions on Audio, Speech and Language Processing **24** (2016), no. 10, 1854–1864.

[64] P. Laffitte, D. Sodoyer, C. Tatkeu, and L. Girin, *Deep neural networks for automatic detection of screams and shouted speech in subway trains*, 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (Shanghai, China), March 2016, pp. 6460–6464.

[65] M. Lagrange, G. Lafay, B. Defreville, and J.J Aucouturier, *The bag-of-frames approach: a not so sufficient model for urban soundscapes*, Journal of the Acoustical Society of America, Acoustical Society of America **138(5)** (2015), 487–492.

[66] Y. LeCun and Y. Bengio, *The handbook of brain theory and neural networks*, ch. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258, MIT Press, Cambridge, MA, USA, 1998.

[67] Y. Lee, D. Han, and H. Ko, *Acoustic Signal Based Abnormal Event Detection in Indoor Environment using Multiclass Adaboost*, IEEE Trans. on Consumer Electronics **59** (2013), no. 3, 615–622.

[68] B. Lei and M. Mak, *Sound-event partitioning and feature normalization for robust sound-event detection*, Int. Conf. on Digital Signal. Processing (Hong Kong), Aug. 20-23 2014, pp. 389–394.

[69] H. Lei and B. Sun, *A study on the dynamic time warping in kernel machines*, 2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System, Dec 2007, pp. 839–845.

[70] J. Li, W. Dai, F. Metze, S. Qu, and S. Das, *A comparison of deep learning methods for environmental sound detection*, 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (New Orleans, LA, USA), vol. abs/1703.06902, 2017.

[71] R. P. Lippmann, *Pattern classification using neural networks*, IEEE Communications Magazine **27** (1989), no. 11, 47–50.

[72] R.P. Lippmann, *From statistics to neural networks: Theory and pattern recognition applications*, ch. Neural Networks, Bayesian a posteriori Probabilities, and Pattern Classification, pp. 83–104, Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

[73] Bruce T. Lowerre, *The harpy speech recognition system.*, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1976, AAI7619331.

[74] R. Lu, Z. Duan, and C. Zhang, *Metric learning based data augmentation for environmental sound classification*, IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (New Paltz, NY), Oct. 2017.

[75] Warren S. McCulloch and Walter Pitts, *A logical calculus of the ideas immanent in nervous activity*, The bulletin of mathematical biophysics **5** (1943), no. 4, 115–133.

[76] I. McLoughlin, H. Zhang, Z. Xie, Y. Song, and W. Xiao, *Robust sound event classification using deep neural networks*, IEEE/ACM Trans. on Audio, Speech, and Language Processing **23** (2015), no. 3, 540–552.

[77] A. Mesaros, T. Heittola, A. Eronen, and T. Virtanen, *Acoustic event detection in real life recordings*, 2010 18th European Signal Processing Conference (Aalborg, Denmark), Aug 2010, pp. 1267–1271.

[78] A. Mesaros, T. Heittola, and T. Virtanen, *Tut database for acoustic scene classification and sound event detection*, 2016 24th European Signal Processing Conference (EUSIPCO), Aug 2016, pp. 1128–1132.

[79] Annamaria Mesaros, Toni Heittola, and Tuomas Virtanen, *Metrics for polyphonic sound event detection*, Applied Sciences **6** (2016), no. 6, article number 162.

[80] Y. Miao, *Kaldi+pdnn: Building dnn-based ASR systems with kaldi and PDNN*, CoRR **abs/1401.6984** (2014).

[81] Tomas Mikolov, Martin Karafiat, and Lukas Burget, *Recurrent neural network based language model*.

[82] S. Mun, S. Shon, W. Kim, D. K. Han, and H. Ko, *Deep neural network based learning and transferring mid-level audio features for acoustic scene classification*, 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), March 2017, pp. 796–800.

[83] K. Nakadai, D. Matsuura, H. G. Okuno, and H. Tsujino, *Improvement of recognition of simultaneous speech signals using av integration and scattering theory for humanoid robots*, Speech Communication **44** (2004), no. 1, 97 – 112, Special Issue on Audio Visual speech processing.

[84] R. M. Neal, *Connectionist learning of belief networks*, Artificial Intelligence **56** (1992), no. 1, 71–113.

[85] R. M. Neal and G. E. Hinton, *Learning in graphical models*, ch. A view of the EM algorithm that justifies incremental, sparse and other variants, pp. 335–368, Dordrecht: Kluwer Academic Press, 1998.

[86] Andrew Y. Ng and Michael I. Jordan, *On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes*, Advances in Neural Information Processing Systems 14 (T. G. Dietterich, S. Becker, and Z. Ghahramani, eds.), MIT Press, 2002, pp. 841–848.

[87] K. Noda, Y. Yamaguchi, K. Nakadai, H.G. Okuno, and T. Ogata, *Audio-visual speech recognition using deep learning*, Applied Intelligence **42** (2015), no. 4, 722–737.

[88] S. Ntalampiras, I. Potamitis, and N. Fakotakis, *On acoustic surveillance of hazardous situations*, IEEE Int. Conf. on Acoustics, Speech and Signal Processing (Taipei, Taiwan), Apr. 19-24 2009, pp. 165–168.

[89] S. J. Pan and Q. Yang, *A survey on transfer learning*, IEEE Transactions on Knowledge and Data Engineering **22** (2010), no. 10, 1345–1359.

[90] G. Parascandolo, H. Huttunen, and T. Virtanen, *Recurrent neural networks for polyphonic sound event detection in real life recordings*, 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (Shanghai, China), March 2016.

[91] Geoffroy Peeters, Bruno L. Giordano, Patrick Susini, Nicolas Misdariis, and Stephen McAdams, *The timbre toolbox: Extracting audio descriptors from musical signals*, The Journal of the Acoustical Society of America **130** (2011), no. 5, 2902–2916.

[92] Geoffroy Peeters and Xavier Rodet, *Automatically selecting signal descriptors for sound classification*, 2002.

[93] A. Pennisi, D.D. Bloisi, and L. Iocchi, *Online real-time crowd behavior detection in video sequences*, Computer Vision and Image Understanding **144** (2016), 166 – 176, Individual and Group Activities in Video Event Analysis.

[94] H. Phan, M. Maas, R. Mazur, and A. Mertins, *Random regression forests for acoustic event detection and classification*, IEEE/ACM Trans. on Audio, Speech, and Language Processing **23** (2015), no. 1, 20–31.

[95] J. Pohjalainen, P. Alku, and T. Kinnunen, *Shout detection in noise*, IEEE Int. Conf. on Acoustics, Speech and Signal Processing (Prague, Czech Republic), 2011, pp. 4968 – 4971.

[96] J. Pohjalainen, T. Raitio, and P. Alku, *Detection of shouted speech in the presence of ambient noise*, Interspeech (Florence, Italy), Aug. 28-31 2011, pp. 2621–2624.

[97] A. r. Mohamed, G. E. Dahl, and G. Hinton, *Acoustic modeling using deep belief networks*, IEEE Transactions on Audio, Speech, and Language Processing **20** (2012), no. 1, 14–22.

[98] A. r. Mohamed, G. Hinton, and G. Penn, *Understanding how deep belief networks perform acoustic modelling*, 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), March 2012, pp. 4273–4276.

[99] L. R. Rabiner, *A tutorial on hidden markov models and selected applications in speech recognition*, Proceedings of the IEEE **77** (1989), no. 2, 257–286.

[100] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review (1958), 65–386.

[101] J.-L. Rouas, J. Louradour, and S. Ambellouis, *Audio events detection in public transport vehicle*, Intelligent Transportation Systems Conference (Toronto, Ont., Canada), Sept. 2006, pp. 733–738.

[102] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, Nature **323** (1986), 533–536.

[103] _____, *Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1*, ch. Learning Internal Representations by Error Propagation, pp. 318–362, MIT Press, Cambridge, MA, USA, 1986.

[104] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, *Convolutional, long short-term memory, fully connected deep neural networks*, 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), April 2015, pp. 4580–4584.

[105] H. Sakoe and S. Chiba, *Dynamic programming algorithm optimization for spoken word recognition*, IEEE Transactions on Acoustics, Speech, and Signal Processing **26** (1978), no. 1, 43–49.

[106] J. Salamon and J. P. Bello, *Feature learning with deep scattering for urban sound analysis*, European Signal Processing Conference (Nice, France), Aug. 30 - Sept. 4 2015, pp. 729–733.

[107] V. Saligrama and Z. Chen, *Video anomaly detection based on local statistical aggregates*, 2012 IEEE Conference on Computer Vision and Pattern Recognition (Providence, RI, USA), June 2012, pp. 2112–2119.

[108] Jürgen Schmidhuber, *Deep learning in neural networks: An overview*, Neural Networks **61** (2015), 85 – 117.

[109] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research **15** (2014), 1929–1958.

[110] R. Stiefelhagen, K. Bernardin, R. Bowers, R.T Rose, M. Michel, and J. Garofolo, *The clear 2007 evaluation*, National Institute of Standards and Technology, 2007.

[111] B. L. Sturm, *Alexander lerch: An introduction to audio content analysis: Applications in signal processing and music informatics*, Computer Music Journal **37** (2013), no. 4, 90–91.

[112] A. Temko, C. Nadeu, D. Macho, R. Malkin, C. Zieger, and M. Omologo, *Computers in the human interaction loop*, ch. Acoustic Event Detection and Classification, pp. 61–73, Springer London, London, 2009.

[113] G. Valenzise, L. Gerosa, M. Tagliasacchi, E. Antonacci, and A Sarti, *Scream and gunshot detection and localization for audio-surveillance systems*, IEEE Conference on Advanced Video and Signal Based Surveillance (London,UK), Sept 2007, pp. 21–26.

[114] X. Valero and F. Alías, *Gammatone wavelet features for sound classification in surveillance applications*, European Signal Processing Conference (Bucharest, Romania.), Aug. 27-31 2012, pp. 1658–1662.

[115] S. Venugopalan, M. Rohrbach, J. Donahue, R. Mooney, T. Darrell, and K. Saenko, *Sequence to sequence – video to text*, IEEE International Conference on Computer Vision (ICCV), Dec 2015, pp. 4534–4542.

[116] T. Virtanen, A. Mesaros, T. Heittola, M.D Plumbley, P. Foster, E. Benetos, and M. Lagrange, *Proceedings of the detection and classification of acoustic scenes and events 2016 workshop (dcase2016)*, Tampere University of Technology. Department of Signal Processing, 2016.

[117] J. Wang, C. Lin, B. Chen, and M. Tsai, *Gabor-based nonuniform scale-frequency map for environmental sound classification in home automation*, IEEE Trans. on Automation Science and Engineering **11** (2014), no. 2, 607–613.

[118] Y. Wang and F. Metze, *A first attempt at polyphonic sound event detection using connectionnist temporal classification*, IEEE Int. Conf. on Acoustics, Speech and Signal Processing (New Orleans, LA, USA), March. 5-9 2017.

[119] D. Wei, J. Li, P. Pham, S. Das, and S. Qu, *Acoustic scene recognition with deep neural networks (DCASE challenge 2016)*, Tech. report, DCASE2016 Challenge, September 2016.

[120] Paul Werbos, *Backpropagation through time: what it does and how to do it*, **78** (1990), 1550 – 1560.

[121] X. Wu, H. Gong, P. Chen, Z. Zhong, and Y. Xu, *Surveillance robot utilizing video and audio information*, J. of Intelligent and Robotic Systems **55** (2009), no. 4, 403–421.

[122] Zhengzheng Xing, Jian Pei, and Eamonn Keogh, *A brief survey on sequence classification*, SIGKDD Explor. Newsl. **12** (2010), no. 1, 40–48.

[123] G. Xiong, X. Wu, Y. L. Chen, and Y. Ou, *Abnormal crowd behavior detection based on the energy model*, 2011 IEEE International Conference on Information and Automation (Trieste, Italy), June 2011, pp. 495–500.

[124] S.J. Young and L.L. Chase, *Speech recognition evaluation: a review of the u.s. csr and lvcsr programmes*, Computer Speech and Language **12** (1998), no. 4, 263 – 279.

[125] G. P. Zhang, *Neural networks for classification: a survey*, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) **30** (2000), no. 4, 451–462.

[126] X.-L. Zhang and J. Wu, *Deep belief networks based voice activity detection*, IEEE Trans. on Audio, Speech, and Language Processing **21** (2013), no. 4, 697–710.

[127] Xi Zhou, Xiaodan Zhuang, Ming Liu, Hao Tang, Mark Hasegawa-Johnson, and Thomas Huang, *Multimodal technologies for perception of humans: International evaluation workshops clear 2007 and rt 2007, baltimore, md, usa, may 8-11, 2007, revised selected papers*, ch. HMM-Based Acoustic Event Detection with AdaBoost Feature Selection, pp. 345–353, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[128] R. Zouaoui, R. Audigier, S. Ambellouis, F. Capman, H. Benhadda, S. Joudrier, D. Sodoyer, and T. Lamarque, *Embedded security system for multi-modal surveillance in a railway carriage*, SPIE security and defence (Toulouse, France), vol. 9652, Sept. 21-24 2015, pp. Paper N 9652–11.