TAMPERE UNIVERSITY OF TECHNOLOGY

GIAMBATTISTA PARASCANDOLO

RECURRENT NEURAL NETWORKS FOR POLYPHONIC SOUND EVENT DETECTION

Master of Science Thesis

Examiners: Tuomas Virtanen, Heikki Huttunen
Examiners and topic approved by the Faculty of Computing and Electrical Engineering

on 12 August 2015

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Signal Processing
PARASCANDOLO, GIAMBATTISTA: Recurrent neural networks for polyphonic sound event detection
Master of Science Thesis, 66 pages
November 2015
Major: Signal Processing
Minor: Learning and Intelligent Systems
Examiners: Tuomas Virtanen, Heikki Huttunen
Keywords: Sound event detection, recurrent neural network, LSTM, deep learning

The objective of this thesis is to investigate how a deep learning model called recurrent neural network (RNN) performs in the task of detecting overlapping sound events in real life environments. Examples of such sound events include dog barking, footsteps, and crowd applauding. When several sound sources are active simultaneously, as it is often the case in everyday contexts, identifying individual sound events from their polyphonic mixture is a challenging task. Other factors such as noise and distortions contribute to making even more difficult to explicitly implement a computer program to solve the detection task.

We present an approach to polyphonic sound event detection in real life recordings based on a RNN architecture called bidirectional long short term memory (BLSTM). A multilabel BLSTM RNN is trained to map the time-frequency representation of a mixture signal consisting of sounds from multiple sources, to binary activity indicators of each event class. Our method is tested on two large databases of recordings, both containing sound events from more than 60 different classes, and in one case from 10 different everyday contexts. Furthermore, in order to reduce overfitting we propose to use several data augmentation techniques: time stretching, sub-frame time shifting, and block mixing.

The proposed approach outperforms the previous state-of-the-art method, despite using half of the parameters, and the results are further largely improved using the block mixing data augmentation technique. Overall, for the first dataset our approach reports an average $F1$-score of 65.5% on 1 second blocks and 64.7% on single frames, a relative improvement over previous state-of-the-art approach of 6.8% and 15.1% respectively. For the second dataset our system reports an average $F1$-score of 84.4% on 1 second blocks and 85.1% on single frames, a relative improvement over the baseline approach of 38.4% and 35.9% respectively.

PREFACE

Before you lies the thesis "Recurrent neural networks for polyphonic sound event detection", the result of a research about machine learning applied to machine hearing. This project was undertaken at the Audio Research Group, in the Signal Processing department of Tampere University of Technology, from February to November 2015.

I would like to extend my deepest thanks to my supervisors Prof. Tuomas Virtanen and Dr. Eng. Heikki Huttunen for our weekly meetings and their excellent guidance during this process. They gave me freedom and encouragement to explore, and I am very glad our collaboration did not end here. I want to thank Emre Cakir for providing the results for the feedforward neural network on the artificial mixtures dataset; I wish to acknowledge CSC — IT Center for Science, Finland, for computational resources.

The last two years in Finland have been a wonderful time. For this I would like to thank all my friends from all over the world that are (or have been) here in Tampere. A special thanks goes to the pals of the board games club for the amusing time spent together. I also want to thank Prof. Massimo Picardello for recommending TUT to me for my Master's degree, without his good advice I would have missed such a great experience.

Finally, I would like to thank my wonderful parents, who taught me the value of learning and thinking, who always showed me unwavering love, and to whom this work is dedicated.

Giambattista Parascandolo
23 November 2015

CONTENTS

TERMS AND DEFINITIONS

ANN	Artificial neural network
AMD	Artificial mixtures dataset
ASR	Automatic speech recognition
BLSTM	Bidirectional long short-term memory
BP	Backpropagation
BPTT	Backpropagation through time
CE	Cross entropy
DNN	Deep neural network
FFT	Fast Fourier transform
FNN	Feedforward neural network
GMM	Gaussian mixture model
HF	Hessian free
HMM	Hidden Markov model
LSTM	Long short-term memory
ML	Machine learning
NMF	Non-negative matrix factorization
NN	Neural network
ReLU	Rectified linear unit
RLRD	Real life recordings dataset
RMSE	Root mean squared error
RNN	Recurrent neural network
SED	Sound event detection
SGD	Stochastic gradient descent
STFT	Short time Fourier transform
SVM	Support vector machine

1. INTRODUCTION

Sensors allow computers to perceive the world as we do, but not to understand it: with cameras they record images and video, but they do not see; with microphones they record sounds, but they do not hear. One of the goals of artificial intelligence is to make computers able to recognize the content of what they receive as input.

Many complex tasks such as recognizing objects or faces in a picture, words in a speech, or sound events in an environment, come naturally to human beings. Our proficiency is to be attributed to our brain, a powerful machine that has adapted to live in our world through millions of years of evolution, and that further modifies itself by learning from experience. Despite our mastery, it is difficult to formally describe these problems, and, as consequence, it is hard to explicitly implement a solution in a computer program. [1]

The development of a field called *deep learning*—which draws inspiration from the way our brain works—has radically changed the way many of these complex problems are approached. Using artificial neural networks with multiple layers of computation and a large amount of data, a system can be trained—often end-to-end—to discover increasingly more complex features from the raw data, and to classify new unseen observations.

Concerning the task of *machine hearing* [2], automatic speech recognition (ASR) has been a very active area of research which has received considerable attention [3] and where artificial neural networks have substantially advanced the state-of-the-art in the recent years [4; 5]. On the contrary, the detection of more general sound events from real life environments—such as a dog barking, footsteps, crowd

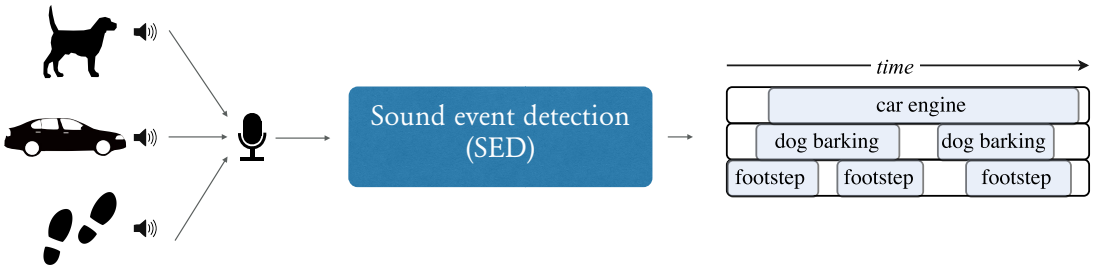

Figure 1.1: An example of polyphonic sound event detection. The systems receives as input an audio recording from a real life environments and outputs, in every short time frame, a label corresponding to each individual source recognized.

applauding—is still a rather new and open research topic [6], known as sound event detection (SED), or acoustic event detection (AED).

A typical real environment is rich in simultaneously active sound events from different sources. Compared to the more simple task of monophonic SED—where only one sound source at the time is active—one of the main challenges for polyphonic SED is to detect the individual sound sources from their mixture. Figure 1.1 illustrates the task of polyphonic SED, which will be the core topic investigated in this thesis.

A robust system for SED in real life recordings would have several applications, including automatic audio indexing [7; 8], which would allow to find where a certain event appears in a large database of recordings or videos; environmental context detection [9], allowing a mobile device to trigger context specific actions; acoustic surveillance [10], such as security and safety applications.

In SED in real life recordings, deep neural networks have recently advanced by a large margin the state-of-the-art [11], improving the results over previous approaches such as Gaussian mixture models with hidden Markov models [12; 13] and non-negative matrix factorization [14; 15]. However, only a certain artificial neural network architecture called *feedforward neural network* (FNN) has been used to approach sound event detection in real life environments. This architecture is not inherently well suited to represent sequential inputs such as audio, video or text, due to the short or null context information available.

A more powerful neural network architecture, called *recurrent neural network* (RNN), has recursive connections that allow information from the past observations to remain inside the network and influence its predictions, making it more adapt to process sequences. Therefore RNNs, contrarily to FNNs, can directly model the sequential information that is naturally present in audio. Their ability to remember past states can avoid the need for tailored postprocessing or smoothing steps that were required in previous works [11]. Moreover, RNNs have already obtained excellent results on complex audio detection tasks, such as ASR [5], onset detection [16] and polyphonic piano note transcription [17].

Motivated by these premises, in this work we investigate the use of recurrent neural networks for polyphonic SED. Moreover, since a large dataset is a key requirement to effectively train deep neural networks, we also examine the effects of different *data augmentation* techniques on the performances, where by data augmentation we refer to methods used to create new instances by manipulating the available data. In order to evaluate our proposed approach, we test it on two large databases of sound events.

This thesis is organized as follows. Chapter 2 and 3 present the theoretical background necessary for the reader to understand the research problems described later

on in the thesis: Chapter 2 focuses on sound event detection and provides a detailed literature review on previous works conducted in this area; Chapter 3 presents artificial neural networks, focusing on recurrent neural networks and the long short-term memory architecture. Chapter 4 describes the process and methodology of our proposed approach to polyphonic sound event detection using multilabel recurrent neural networks. In Chapter 5 we present the experimental set-up and results of our approach tested on two databases of audio recordings. Finally, concluding remarks and propositions for future research are given in Chapter 6.

A large part of the results presented in this thesis have been submitted in [18].

2. SOUND EVENT DETECTION

This chapter introduces the concept of sound event detection and its theoretical background, reviewing previous work done in the field.

2.1 Supervised learning

We often want to make computers able to solve problems whose solutions are difficult to explicitly implement, such as recognizing handwritten digits, transcribing spoken words in a speech, distinguishing the objects in a picture, or predicting if in patient a certain disease will relapse. In all the cases mentioned, we read an input $\mathbf{x}$ and associate one or more labels $\mathbf{y}$. In the field of machine learning, the task of predicting a label $\mathbf{y}$ for a new unseen input $\mathbf{x}$ is called *classification*. A raw input observation might contain redundant information or be too high dimensional to be classified directly, so typically some of its features are selected or extracted first, and then used for classification.

If we have a sufficient amount of labeled examples, *i.e.*, pairs of input and corresponding output $(\mathbf{x}, \mathbf{y})$, we can make a computer *learn* how to classify new unseen data by training it on the known instances (Figure 2.1).

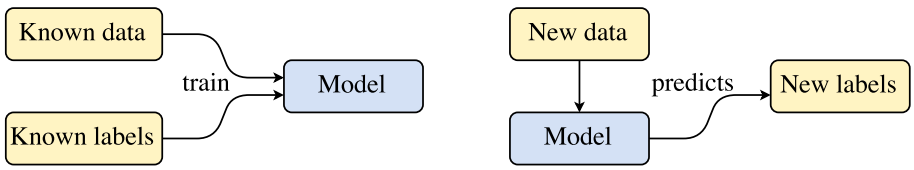

Figure 2.1: Supervised learning workflow.

Supervised learning is the task of inferring a function from labeled training data. Formally, we want to learn a function

$$f : \mathcal{X} \to \mathcal{Y}, \tag{2.1}$$

where $\mathcal{X}$ is an input space and $\mathcal{Y}$ an output space, from a set of N examples of the form $\{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_N, \mathbf{y}_N)\}$, where $\mathbf{x}_n$ is an input vector and $\mathbf{y}_n$ the corresponding target. For an input vector $\mathbf{x}$ we call its predicted output $\hat{\mathbf{y}} = f(\mathbf{x})$. In the context of classification, the goal is to produce a function f that would most accurately determine the class labels of new unseen instances. The function f is referred to as

the *model*.

Formally, for a parametrical model f with parameters $\boldsymbol{\theta}$, we want to minimize a cost function E that measures the distance between all the predictions $\hat{\mathbf{y}}_n$ and the targets $\mathbf{y}_n = f(\mathbf{x}_n)$ in the dataset:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} E(f(\mathbf{x}_n; \boldsymbol{\theta}), \mathbf{y}_n). \tag{2.2}$$

This is known as the minimization of the empirical risk, *i.e.*, the risk of incorrectly classifying a known instance.

However, since the goal of any classification system is to correctly classify new unseen instances, its accuracy cannot be solely evaluated based on the data it was trained on. When a model performs well on known data and poorly on unseen data, it is said to have overfitted to the training data. This typically occurs for very flexible models, which might start to memorize the training instances (*overfitting*) rather than learn the true underlying structure of the data (*generalization*). A typical approach to ensure that the model has generalized is to evaluate it on unseen data whose labels are known. For this reason the original labeled data is usually split into three parts: the *training* set, *i.e.*, the data used in training to learn the parameters of the model; the *validation* set, shown to the model but not used for training, it monitors when overfitting starts to arise; the *test* set, only shown to the model when the training is complete, is used to evaluate its accuracy on new data.

Regularization techniques help to reduce overfitting by penalizing models with extreme parameters or by using an ensemble of models. One of the most widely used regularization techniques in learning models is *early stopping*, which halts the training once the validation error starts to increase. Other popular techniques include penalty terms (*e.g.* L_1 and L_2 regularization), ensemble learning and data augmentation (discussed in Section 3.5).

The label vectors $\mathbf{y}$ can be of three types, leading to three kinds of classification tasks, *i.e.*, binary, multiclass and multilabel classification. When the input observations belong to exactly one of two separate sets, *i.e.*, $\mathbf{y}$ is a scalar $y \in \{-1, 1\}$, the task is called *binary classification*. An example of binary classification task is determining if an email is spam or not.

When each input instance belongs to exactly one of K classes ($K > 2$), *i.e.*, $y \in \{1, ..., K\}$, the task is known as *multiclass classification*. In one-hot encoding vector notation, to mark that a certain data point belongs to class k we write for $\mathbf{y}$ that $y_k = 1$ and $y_j = 0$, $\forall j \neq k$, where y_k is the k^{th} entry of $\mathbf{y}$. Examples of multiclass classification include automatic speech recognition (ASR), where each audio segment is classified as one word present in the dictionary.

In the most general case, an input instance might belong to none, one or several

among the K classes at the same time, *i.e.*, $\mathbf{y} \in \mathcal{Y} = \{0, 1\}^K$. This scenario is known as *multilabel classification*. Examples include polyphonic music transcription, in which multiple notes are played at the same time, and document classification, where each document in a corpus may concern several topics. Multilabel classification can be particularly difficult, and its main challenges are the following:

High cardinality of the power set The upper bound of the number of possible label combinations is the cardinality of the power set of the set of labels

$$\sum_{k=0}^{K} \binom{K}{k} = 2^K, \tag{2.3}$$

compared to the cardinality K of the label set in multiclass classification, and cardinality 2 in binary classification.

Broader concept of generalization In binary and multiclass classification tasks, the training data usually consists of at least a small number of instances from each class. Therefore, at test time, the model needs to recognize a known class in a new instance. In multilabel classification, since the output space is very high-dimensional, it is likely that there are not any instances for several classes combinations. A model should be able to generalize one step further, potentially recognizing in a new data vector even an unseen class combination.

Evaluation measure Evaluating the results of a binary or multiclass classification task is straightforward, since there is always a single right answer—*i.e.*, the correct class—and the model can either correctly detect it or not. In multilabel classification each sample belongs to k classes, where $0 \leq k \leq K$. The evaluation measure should take into account how many classes were correctly detected, incorrectly detected, missed, and how to mutually weigh them.

Label dependencies In multilabel classification tasks there are often dependencies among labels [19], *i.e.*,

$$p(\mathbf{y}) \neq \prod_{k=1}^{K} p(y_k) \qquad p(\mathbf{y}|\mathbf{x}) \neq \prod_{k=1}^{K} p(y_k|\mathbf{x}). \tag{2.4}$$

Both discovering and modeling these dependencies can be very difficult. Ignoring the possible correlations can simplify the problem but might negatively affect performance. *Binary relevance* (BR), considered to be the baseline approach for multilabel classification, ignores the dependencies by reducing the problem to K independent binary classification tasks [20].

2.2 Polyphonic sound event detection

Sound event detection (SED), also known as acoustic event detection (AED), deals with the identification of sound events in audio recordings. By *sound event* we refer to a segment of audio produced by a sound source, which can be categorized with a label such as *footsteps, phone ringing,* or *door slamming*. The goal is to estimate start and end times of sound events, and to assign a label for each event. As illustrated in Figure 2.2, a SED system receives audio as input, processes the signal (feature extraction), detects the sound events (classification), and outputs their labels accordingly.

In a simple case, a system is trained to operate under conditions where only one sound source is emitting at a certain time, and the goal is to recognize which sound source is emitting, when the event begins and when it ends. SED in single-source environment is called *monophonic* detection, which has been the major area of research in this field [6; 21; 22]. A detection problem is considered to be monophonic also in cases where only one sound is prominent with respect to the others, and a single label is expected.

However, in a typical real environment it is uncommon to have only a single sound source emitting at a certain point in time; it is more likely that multiple sources are emitting simultaneously, thus resulting in an additive combination of sounds. For example, an audio recording collected at a restaurant table might contain sounds from cutlery, speech, footsteps from the waiters and a soft background music. A depiction of this example can be seen in Figure 2.3. Due to the presence of multiple and overlapping sounds, this problem is known as *polyphonic* detection, and the goal of such a SED system is to recognize for each sound event its category, and its beginning and ending. This task is much more challenging than the monophonic detection problem, both because the sounds are overlapping and because the number of sources emitting at any given moment—*i.e.*, its *polyphony* level—is unknown and potentially large.

As in all classification tasks, the system receives an input and is expected to

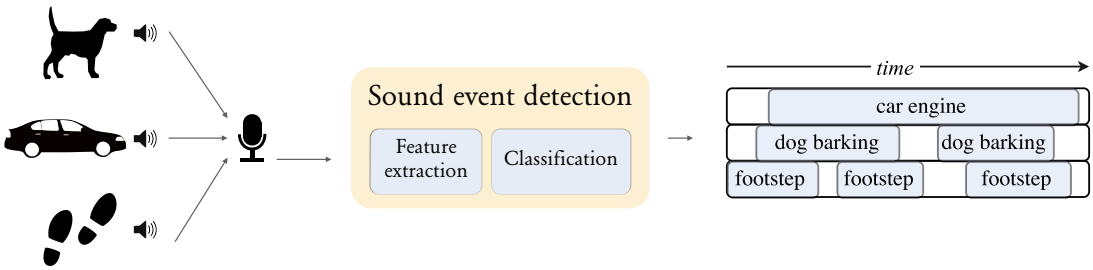

Figure 2.2: In SED, a system is expected to recognize the sources of the sounds present in a recording over time.

produce as output one or more labels for such input. The monophonic detection is a multiclass classification task, while the polyphonic detection is multilabel. Each label corresponds to one sound source or category; for example at time t in Figure 2.3 we would expect the labels *speech* and *music* to be output from the system.

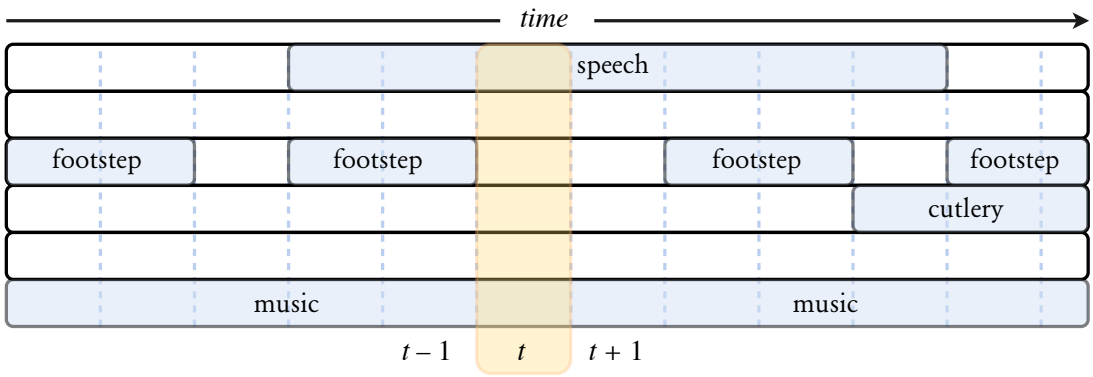

Figure 2.3: An illustration of class activities in a polyphonic sound event detection scenario. At time t the events *speech* and *music* are active.

2.3 Audio features

The input to a SED system is typically a raw digital audio recording, in its standard format of sampled and quantized analog amplitude of air pressure variation (pulse code modulated, PCM). This simple one-dimensional representation of an audio signal can be difficult to interpret directly: two similar sound sources might emit sounds that are perceived as almost identical but produce very different waveforms. Factors such as noise or phase disparities can drastically change the shape of the amplitude signal but might have a small effect on the perceived sound.

For two given acoustic events, the magnitude of their frequency content over time is a more robust feature to reveal their similarities. This representation of an acoustic signal, widely used in the field of audio processing, is known as time-frequency representation. Each sound source has a characteristic pattern of spectral magnitudes over time, which can be detected for classification purposes (Figure 2.4).

For this reason a typical first step in a SED system is to process the audio signal, in order to convert it to a time-frequency domain representation. Due to the *uncertainty principle* [23] this representation has necessarily a coarser time resolution than the original signal; this does not represent an issue, since the time resolution of a PCM signal is usually quite high (typically 44.1 kHz or more) and the goal in SED is to recognize sounds in time frames that are several milliseconds long.

Most of the approaches for SED rely on time-frequency representations of the signal—*e.g.* through short-time Fourier transform or wavelet transform—possibly filtered with filterbanks that mimic the perception of human hearing, such as mel or

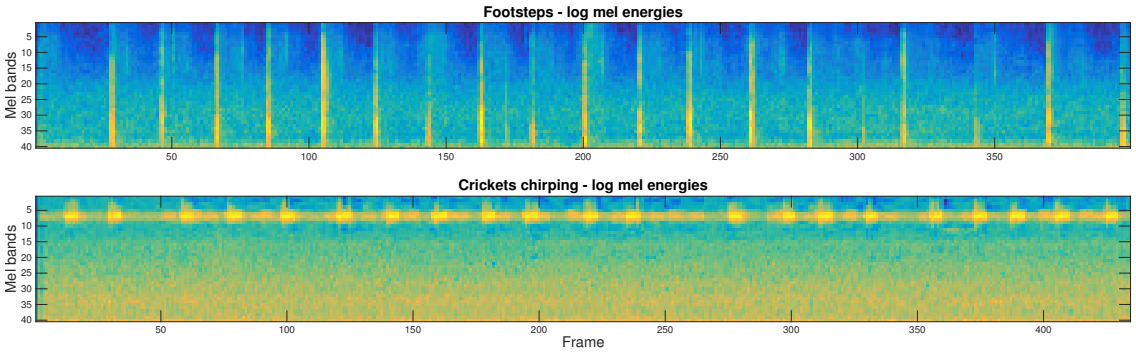

Figure 2.4: A time-frequency representation of two recordings: footsteps and crickets chirping. Both sounds are intermittent, but they have clear and distinct magnitude patterns.

gammatone [6]. Some approaches try to model the signal in its new representation, some others extract features that are discriminant for classification. We describe here three feature representations relevant for understanding the work presented in the following chapters. Alternative features and representations typically used include MPEG-7 audio descriptors [24], wavelet transforms [25] and gammatone filterbanks [26].

Short-time Fourier transform The most common method used in signal processing to transform a signal to a time-frequency representation is the short-time Fourier transform (STFT). The original signal is first windowed into short, overlapping time-frames. Fast Fourier transform (FFT) is computed for each frame, assuming that the signal is stationary in short windows. The concatenation along time of the extracted frequency magnitudes, also known as *spectrogram* (Figure 2.5), represents the transformed signal.

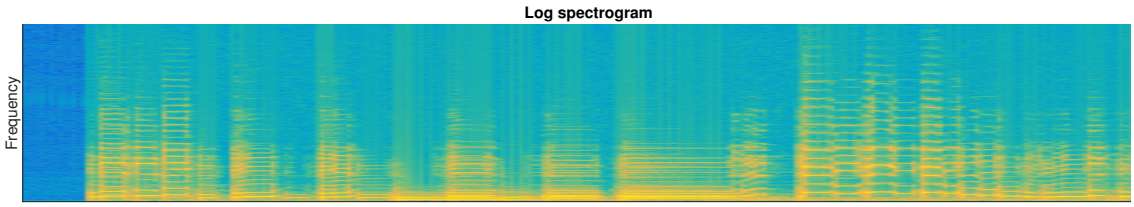

Figure 2.5: Logarithmic magnitude spectrogram of a short recording.

Mel energies In order to mimic the human ear's perception of frequencies, which is roughly linear below 1 kHz and logarithmic above, a suitable filterbank can be applied to the frequency magnitudes extracted via the STFT. A widely used transformation is from Hertz to *mel*, a scale where perceptually equal pitches are equally distant [27]. The mel scale m is defined as a function of the frequency in Hertz f_{Hz}

as

$$m(f_{\text{Hz}}) \equiv 2595 \log_{10}\left(1 + \frac{f_{\text{Hz}}}{700}\right). \tag{2.5}$$

The conversion to *mel energies* is obtained by filtering the magnitude spectrum with a filterbank of triangular filters, shown in Figure 2.6. These filters are equally spaced along the mel scale. Each triangular filter identifies a band and has its edges

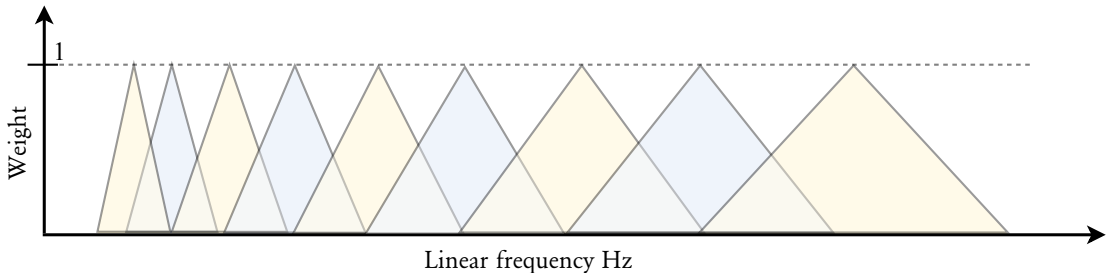

Figure 2.6: The triangular mel filterbank.

on the central frequency of its neighboring filters, hence the filters are wider as frequency grows. The number of bands can be varied to obtain a coarser or finer resolution, typical values used in literature are 20, 40 and 80 bands. Computing the logarithm of the energies obtained, producing so-called *log mel energies*, is a popular choice to mimic human perception of sound intensity, which is approximately logarithmic. Figure 2.7 depicts a log mel spectrogram.

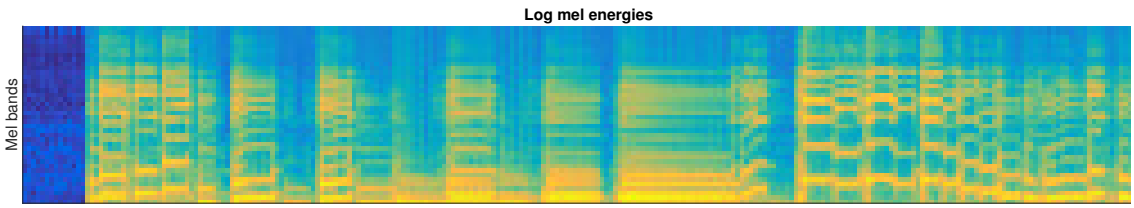

Figure 2.7: Log mel energies of the same recording used in Figure 2.5.

Mel frequency cepstral coefficients Adjacent bands in a mel energy frame are usually highly correlated. To further reduce the number of parameters, it is possible to decorrelate the log mel spectral vector by computing its discrete cosine transform (DCT), and then extracting the first coefficients of the DCT. These coefficients, known as *mel-frequency cepstral coefficients* (MFCC) [28], are widely used as features in audio processing [29; 30]. The first coefficient is typically discarded, since it contains the frame's average log energy and therefore does not provide information about the spectral envelope. The second to twelfth coefficients—or even more—are frequently used as a feature vector (Figure 2.8). Possibly, also the MFCC's first and

second time derivative—called *delta* and *acceleration* coefficients respectively—are concatenated to the feature vector.

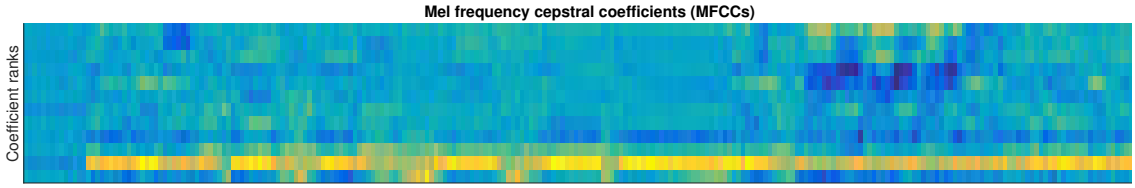

Figure 2.8: Mel frequency cepstral coefficients of the same recording used in Figure 2.5 and Figure 2.7.

When two sound sources are emitting at the same time, the resulting signal is the additive combination of the two waveforms, *i.e.*, $s_{\text{mix}(1,2)}(t) = s_1(t) + s_2(t)$. Due to the linearity of the Fourier transform $\mathscr{F}$, the transform of the mixture signal is equal to the sum of the transform of the individual signals s_1 and s_2

$$\mathscr{F}\left\{s_{\text{mix}(1,2)}(t)\right\}(\omega) = \mathscr{F}\left\{s_1(t)\right\}(\omega) + \mathscr{F}\left\{s_2(t)\right\}(\omega), \tag{2.6}$$

where the variable ω represents frequency. However, this does not hold for the magnitude spectrograms, due to the non-linearity of the absolute value function computed on the complex spectrum obtained from the Fourier transform. Despite this theoretical limitation, in practice linearity is widely assumed in many techniques applied to audio processing such as independent component analysis (ICA) [31; 32] and non-negative matrix factorization (NMF) [33; 34], under the assumption that the phases are independently and randomly distributed .

Each sound source has a typical pattern of magnitude spectrum in the time-frequency domain. A polyphonic SED needs to recognize each sound source from the (imperfect) superposition of magnitude spectra from different sources. In principle, for combinations of sounds present in training data a system might learn to map the learned combination of features to the corresponding combination of classes without actually recognizing the individual components. However, for unseen combinations of sounds the system needs to be able to discern the individual sound sources from the mixture; this is one of the main challenges of polyphonic SED.

2.4 Previous work

Most state-of-the-art monophonic SED systems assume that each sound event occurs in isolation. For this reason, the majority of these systems are not suitable for detecting multiple overlapping sounds. For a comprehensive treatment and for references to the extensive literature on monophonic SED—which is not the scope of this work—one may refer to [6].

Polyphonic SED has been approached with different methodologies. Initial approaches include traditional methods for speech recognition, such as the use of mel frequency cepstral coefficients (MFCC) as features, with Gaussian mixture models (GMM) combined with hidden Markov models (HMM) [12; 13].

Another approach is to take advantage of the robustness of monophonic SED systems, by creating a separate class for each combination of classes that appears in training. An example of this approach using hierarchical support vector machines (SVMs) is [35]. First an SVM determines if the input is either an isolated event or a combination of events, and in the latter case the input is fed to a second SVM to determine the individual sources in the combination. However this approach requires training data for every possible combinations of classes and for different degrees of overlapping, which is often not available for real life recordings.

A different type of methodology consists of extracting and matching the sounds in the input to templates in a dictionary of sounds. This can be achieved through sound source separation techniques, such as non-negative matrix factorization (NMF) on time-frequency representations of the signals. NMF has been used in [14] and [36] to pre-process the signal, creating a dictionary from single events, and later in [13] and [37] directly on the mixture, without learning from isolated sounds. The work in [37] was extended in [15] making learning feasible for long recordings by reducing the dictionary size.

Non-frame based approaches include spectrogram analysis with image processing techniques such as local spectrogram features (LSF) [6] combined with object recognition methods. The work in [38] proposes to learn LSF for each sound event in isolation, and at test time to detect the sound events from a mixture using generalized Hough transform (GHT).

Neural networks have also been used as classifiers in SED. The use of feedforward neural networks (FNN) trained on mel energies or MFCCs of the mixture of sounds is attested in [11]. In that work, a FNN in the form of time-windowed multi layer perceptron (MLP) obtained state-of-the-art results in polyphonic SED for real life recordings. The major advantages of this system are that it neither requires isolated sounds for training nor information about the number of overlapping sounds. The method presented in [11] will serve as a baseline system to evaluate the approach proposed in this thesis.

In conclusion, the detection of sound events in a mixture of overlapping sounds is still a rather new and open research topic, especially concerning recordings in real life environments.

3. NEURAL NETWORKS

This background chapter reviews neural networks, with a focus on recurrent neural networks, and presents the concept of data augmentation.

In computer science, and machine learning in particular, artificial neural networks (ANNs), also known simply as neural networks (NNs), are computing systems that process information by their dynamic state response to external inputs [39]. In its fundamental structure a NN is a network of interconnected nodes, simple processing units known as *neurons*; weighted connections join the neurons and scale the strength of the transmitted signals, representing the synapses in the brain. Its resemblance with the brain is marked by two aspects, *i.e.*, that knowledge is acquired through a learning process, and that it is stored in the connections between neurons [40, p.2].

Neural networks are employed in a wide variety of tasks in machine learning, such as pattern classification, regression and time-series prediction. Neurons, layers and activation functions are common components of most NNs despite the variety of architectures introduced over the years.

Neuron A neuron is the basic unit of a NN. Each neuron receives inputs through its incoming weighted connections from other neurons and possibly itself. For each connection the input is received as the transmitted signal multiplied by the connection weight w. The sum of the weighted received signals and a bias term is computed, then passed through an activation function and finally output through an outgoing weighted connection (Figure 3.1).

Layer Neurons in the network are grouped into layers. There is one input layer, a variable number of hidden layers and one output layer. Each layer receives inputs

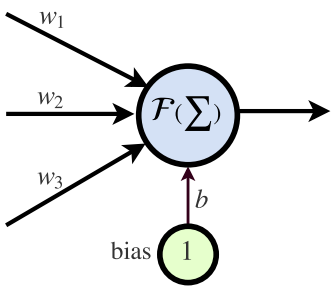

Figure 3.1: A simple model of a neuron.

from the preceding layer (and possibly itself) and delivers outputs to the following. The input layer is composed of D nodes, where D is the dimensionality of the input data; each node reads one of the components of an input vector $\mathbf{x} \in \mathcal{X}$ and outputs it to the following layer's neurons. Hidden layers are between the input and output layer, and perform intermediate computations of the network. A NN is called a deep neural network (DNN) when it has several stacked hidden layers, where the depth increases with the number of hidden layers. The consequences of using deep architectures will be discussed more in detail in Section 3.1. The output layer in classification tasks typically consists of a neuron for each class. For a given output neuron k, its computed value is usually interpreted—eventually after normalization in the range $[0, 1]$—as the posterior probability for the input to belong to class k. All the incoming connections for the neurons in one layer are concatenated to form a matrix $\mathbf{W}$. Together with the bias vectors $\mathbf{b}$, the weight matrices for all layers $\mathbf{W}$ represent the parameters $\boldsymbol{\theta}$ of the model.

Activation function An activation function scales the activation of a neuron into an output signal. Any function could serve as an activation function, however there are few activation functions—shown in Figure 3.2—commonly used in NNs:

- *Logistic function*

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

  The logistic function is a smooth approximation of the step function used in the early stages of neural networks. The output is in the range $[0, 1]$, therefore it is typically used for output neurons in classification tasks. It is also commonly referred to as *sigmoid* function.

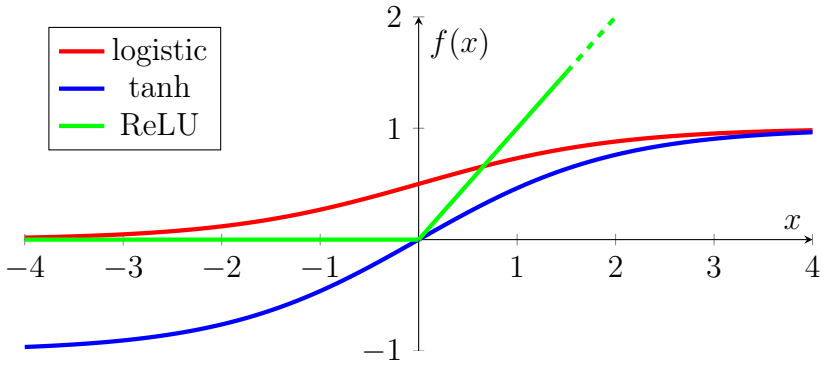

Figure 3.2: The logistic function, hyperbolic tangent (tanh) and rectified linear unit (ReLU) activation functions. It can be seen that while the logistic and tanh both saturate, the ReLU grows unbounded for positive values of x.

- *Hyperbolic tangent (tanh)*

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.2}$$

  The hyperbolic tangent is a logistic function vertically scaled to output in the range $[-1, 1]$.

- *Rectified linear unit (ReLU).*

$$\text{ReLU}(x) = \max(0, x) \tag{3.3}$$

  The ReLU [41] promotes sparse representations inside the network[1] due to the hard 0 for negative values of x. It also avoids saturation problems and vanishing gradients, two of the major problems that arise in deep networks.

- *Softmax* For a vector $\mathbf{x}$, its softmax is defined for each of its components x_j as

$$\text{softmax}(x_j) = \frac{e^{x_j}}{\sum_m e^{x_m}}, \tag{3.4}$$

  such that $\text{softmax}(x_m) > 0 \ \forall m$ and $\sum_m \text{softmax}(x_m) = 1$ [42]. The softmax is mostly used to normalize the outputs of multiclass classification tasks.

All these activation functions are purposefully nonlinear. This is because nonlinear neural networks are very expressive; they can discover nonlinear classification boundaries and model nonlinear equations.

We can now introduce the most simple type of neural network, the multi layer perceptron (MLP) [43; 44]. In this neural network architecture all the layers are sequentially ordered and each layer only receives input from the previous one (Figure 3.3). For an input vector $\mathbf{x}$, a MLP computes the hidden activation vectors $\mathbf{h}$ and the output $\hat{\mathbf{y}}$ as:

$$\mathbf{h} = \mathcal{F}(\mathbf{W}^{\text{xh}}\mathbf{x} + \mathbf{b}^{\text{h}}) \tag{3.5}$$

$$\hat{\mathbf{y}} = \mathcal{G}(\mathbf{W}^{\text{h}\hat{\text{y}}}\mathbf{h} + \mathbf{b}^{\hat{\text{y}}}) \tag{3.6}$$

where $\mathbf{W}^{\star\star}$ denote the weight matrices connecting two layers, *i.e.*, $\mathbf{W}^{\text{xh}}$ are the weights from input to hidden layer and $\mathbf{W}^{\text{hy}}$ from hidden to output layer, $\mathbf{b}^{\star}$ are the bias vectors, and $\mathcal{F}$ and $\mathcal{G}$ are activation functions. When an activation function is computed for all the neurons in a layer using vector notation, such as in Eq. 3.5 and 3.6, it is always computed element-wise. In case of multiple hidden layers the input to hidden layer $\mathbf{h}^l$ is the output of the previous hidden layer $\mathbf{h}^{l-1}$.

[1] For the advantages of sparsity see [41, p.317]

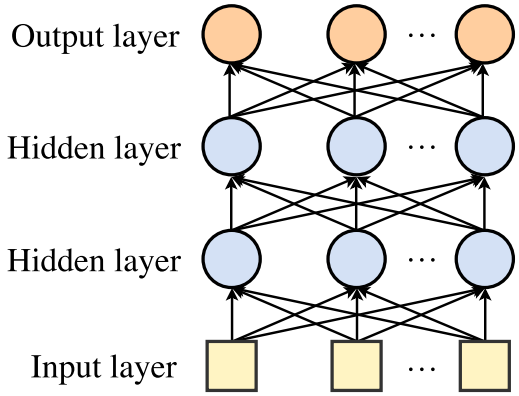

Figure 3.3: A multi layer perceptron with 2 hidden layers. Bias terms are omitted for clarity.

For an input $\mathbf{x}$ a prediction $\hat{\mathbf{y}}$ is computed at the output layer, and compared to the original target $\mathbf{y}$ using a *cost function* $E(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y})$. The network is trained to minimize $E(\mathbf{W}, \mathbf{b}; \mathbf{x}, \mathbf{y})$ for all the input samples $\mathbf{x}$ in the training set, formally:

$$E(\mathbf{W}, \mathbf{b}) = \frac{1}{N} \sum_{n=1}^{N} E(\mathbf{W}, \mathbf{b}; \mathbf{x}_n, \mathbf{y}_n). \tag{3.7}$$

where N is the number of training examples. Since $E(\mathbf{W}, \mathbf{b})$ always depends on $\mathbf{W}$ and $\mathbf{b}$, we will refer to it as E for simplicity.

Generally, *cross entropy* (CE) is used as a cost function to train NNs for classification tasks, with the following equation:

$$\text{CE} = -\frac{1}{N} \sum_{n=1}^{N} \mathbf{y}_n \log \hat{\mathbf{y}}_n + (1 - \mathbf{y}_n) \log(1 - \hat{\mathbf{y}}_n), \tag{3.8}$$

where $\hat{\mathbf{y}}_n$ and $\mathbf{y}_n$ are respectively the prediction and the target output of an instance $\mathbf{x}_n$. Two other cost functions widely used are the sum of squared errors (SSE) and its variation root mean squared error (RMSE):

$$\text{SSE} = \sum_{n=1}^{N} (\mathbf{y}_n - \hat{\mathbf{y}}_n)^2 \qquad \text{RMSE} = \sqrt{\frac{\sum_{n=1}^{N} (\mathbf{y}_n - \hat{\mathbf{y}}_n)^2}{N}}. \tag{3.9}$$

Since NNs are constructed as differentiable operators, they can be trained to minimize the differentiable cost function using *gradient descent* based methods. The intuitive idea is to find the gradient on the error surface for a given weight configuration, and iteratively adjust the weights by moving in the direction of the negative slope. The weights $\mathbf{W}$ and biases $\mathbf{b}$ are generally initialized with random values and then iteratively adjusted during training. An efficient algorithm widely used to compute the gradients for all the weights in the network is the *backpropagation*

algorithm [43; 44], an implementation of the chain rule for partial derivatives along the network.

To illustrate the backpropagation algorithm on a MLP we will use the following notation: $\mathcal{F}$ is an activation function and $\mathcal{F}'$ its first derivative; w_{ji}^l is the weight connecting the i^{th} neuron in layer $l-1$ to the j^{th} neuron in layer l; z_j^l is the weighted input to the j^{th} neuron in layer l, *i.e.*,

$$z_j^l = \sum_i w_{ji}^l h_i^{l-1} + b_j^l; \tag{3.10}$$

h_j^l is the activation of the j^{th} neuron in layer l, *i.e.*, $h_j^l = \mathcal{F}(z_j^l)$; the boldface version of the aforementioned quantities represents the concatenation of the all the respective values in one layer.

Gradient descent can be used to minimize the cost function E, since the network is constructed of differentiable elements. The algorithm requires to compute the derivative of the cost function with respect to each of the weights and bias terms in the network. Once the gradients $\frac{\partial E}{\partial w_{ji}^l}$ and $\frac{\partial E}{\partial b_{ji}^l}$ have been computed, the corresponding weights and biases in the network can be updated by taking a small step towards the negative direction of the gradients, for example using stochastic gradient descent:

$$\Delta w_i(\tau+1) = -\eta \frac{\partial E}{\partial w_i} \tag{3.11}$$

$$w_i(\tau+1) = w_i(\tau) + \Delta w_i(\tau+1), \tag{3.12}$$

where $\Delta w_i(\tau+1)$ is the weight update, η is the *learning rate* and controls the amount of update, and τ is the index of training iterations. The same update rule applies to the bias terms, with b in place of w.

Backpropagation is a technique that allows to efficiently compute the gradients for all the parameters of the network, by propagating backwards the errors at the output layer. First, the backpropagated error δ_j^L is computed for each neuron in the output layer[2], the L^{th} layer:

$$\delta_j^L \equiv \frac{\partial E}{\partial z_j^L} = \frac{\partial E}{\partial h_j^L}\frac{\partial h_j^L}{\partial z_j^L} = \frac{\partial E}{\partial h_j^L}\mathcal{F}'(z_j^L) \tag{3.13}$$

where $\frac{\partial E}{\partial h_j^L}$ depends on the cost function E.

We can then compute the backpropagated errors δ_j^l at the l^{th} layer in terms of

[2]The backpropagated errors δ represent the errors relative to each neuron, and are not to be confused with the overall cost function E [45, p.165].

the backpropagated error δ_j^{l+1} in the next layer as

$$\delta_j^l \equiv \frac{\partial E}{\partial z_j^l} = \sum_i \frac{\partial E}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial z_j^l} = \sum_i w_{ij}^{l+1} \delta_i^{l+1} \mathcal{F}'(z_j^l), \tag{3.14}$$

where we used the equivalence

$$\frac{\partial z_j^{l+1}}{\partial z_j^l} = \frac{\partial}{\partial z_j^l} \sum_i w_{ij}^{l+1} \mathcal{F}(z_j^l) + b_i^{l+1} = \sum_i w_{ij}^{l+1} \mathcal{F}'(z_j^l), \tag{3.15}$$

and by definition

$$\delta_i^{l+1} \equiv \frac{\partial E}{\partial z_i^{l+1}}. \tag{3.16}$$

Finally, we can express the gradients $\frac{\partial E}{\partial w_{ji}^l}$ and $\frac{\partial E}{\partial b_{ji}^l}$ in terms of the error δ_j^l

$$\frac{\partial E}{\partial w_{ji}^l} = \frac{\partial E}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{ji}^l} = h_i^{l-1} \delta_j^l, \tag{3.17}$$

where we used again the definition of δ_j^l and

$$\frac{\partial z_j^l}{\partial w_{ji}^l} = \frac{\partial}{\partial w_{ji}^l} \sum_i w_{ji}^l h_i^{l-1} + b_j^l = h_i^{l-1}. \tag{3.18}$$

For the gradients with respect to the bias terms, using the same procedure for Eq. 3.17 we obtain

$$\frac{\partial E}{\partial b_j^l} = \delta_j^l, \tag{3.19}$$

where the only difference is the h_i^{l-1} term that disappears when computing

$$\frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial}{\partial b_j^l} \sum_i w_{ji}^l h_i^{l-1} + b_j^l = 1. \tag{3.20}$$

Basic stochastic gradient descent might lead to slow convergence. Several techniques such as momentum, Adagrad [46], Adadelta [47], Nesterov accelerated gradient (NAG) [48], RMSprop [49], Adam [50], can greatly improve convergence speed. Here we briefly describe stochastic gradient descent with momentum, and RMSprop, which will be used in the subsequent sections of this work.

Stochastic gradient descent and momentum The technique of momentum [51] can accelerate the gradient descent by accumulating velocity in the direction of steady error reduction across iterations. A new term m, the momentum coefficient,

is added to the weight update equation 3.11 as follows

$$\Delta w_i(\tau+1) = -\eta \frac{\partial E}{\partial w_i} + m\Delta w_i(\tau), \tag{3.21}$$

and typical values of m are in the range of $0.9 \leq m < 1$.

RMSprop RMSprop is an optimizer with a per-parameter adaptive learning rate. It uses a moving average of the magnitude of recent gradients to normalize the current gradients. The normalization is performed over the root mean squared gradients, hence the name RMSprop. The learning rate η is typically called *step rate*, and the running average term $r(\tau)$ is added to the weight update equation:

$$r(\tau) = \gamma r(\tau-1) + (1-\gamma)\left(\frac{\partial E}{\partial w_i}\right)^2 \tag{3.22}$$

$$\Delta w_i(\tau+1) = \frac{-\eta}{\sqrt{r(\tau)}} \cdot \frac{\partial E}{\partial w_i}, \tag{3.23}$$

where γ is the *decay term*—typically set to 0.9—that controls the contribution of new gradients to the running average $r(\tau)$.

It is important to initialize the weights $\mathbf{W}$ and biases $\mathbf{b}$ with positive and negative values close to 0, so that the sigmoidal activation functions operate in their central linear region and thus the propagated gradients are larger [52]. The values are typically drawn randomly and independently from uniform or Gaussian distributions. For the same reason mentioned above, the input features are normalized to have zero mean and unit variance in each dimension.

Several neural network architectures have been introduced over the years. When the connections in the network form cycles, the architecture is referred to as recurrent neural network (RNN), described in Section 3.3 and 3.4. If there are no cycles—such as in the MLP—the network is referred to as feedforward neural network (FNN), described in Section 3.2. Before presenting the details of these different architectures, we briefly introduce the advantages of using deep neural networks.

3.1 Deep learning

Despite the fact that neural networks were invented several decades ago—with the first works on perceptrons dating back to 1950's and multi layer perceptron 1980's—their greatest successes are quite recent, *e.g.* [53; 5]. The major improvements are usually attributed to technological and theoretical factors. On the technological side, faster computers, the use of graphical processing unit (GPU) instead of CPU for training, and the availability of large datasets have made it possible to train

large networks with millions of parameters in reasonable amounts of time. From the theoretical point of view, the main result is that the use of multiple hidden layers exponentially grows the expressiveness of the models [54; 55; 56]. Other significant improvements include unsupervised pre-training (nowadays typically avoided when a large amount of data is available) [57], dropout [58], ReLU [41], and proper random initialization [52].

In classic machine learning, the typical approach to classification is to design discriminative features by hand and then to use a general purpose classifier on top, such as a support vector machine (SVM) [59]. For complex tasks—such as computer vision, ASR and natural language processing—good features that are sufficiently expressive are very difficult to design. Moreover, when the number of features—and thus dimensions—grows, the *curse of dimensionality* makes the available data sparse. Without using distributed representations it is not trivial to generalize to regions of the space that have no training instances, often resulting in poor generalization for complex tasks [60; 1].

Deep learning has changed the way these problems are approached. A deep model has several hidden layers of computations that are used to automatically discover increasingly more complex features and allow their composition. By learning and combining multiple levels of representations, the number of distinguishable regions in a deep architecture grows almost exponentially with the number of parameters, with the potential to generalize to non-local regions unseen in training [55].

These powerful models would be useless if they were not trainable. As previously explained in this chapter, backpropagation can be used to compute the gradients to train these models, updating the weights to minimize the cost function on the training set. Despite the fact that there is no guarantee that the global minimum will be reached, it has been argued that in high dimensional spaces—such as the ones of large and deep neural networks—most of the local minima errors are close to the global minimum error, and that saddle points are mostly responsible for learning stalls [61; 62].

3.2 Feedforward Neural Networks

A FNN is a NN that does not contain any feedback connection, i.e, each layer receives inputs only from the previous layer. Such a network is a directed graph, where information is always traveling forward. There are many architectures of FNN, two of the most used in classification are multi layer perceptrons (MLPs) and convolutional neural networks (CNNs) [63; 64].

Multi Layer Perceptron The multi layer perceptron—presented earlier in this chapter—is the most simple type of FNN. An argument in favor of MLP's expressive

power is given by the *universal approximation theorem* [65]. The theorem states that a MLP with a single hidden layer and sufficient amount of non-linear units can approximate any smooth function on a compact input domain to arbitrary precision. However, the number of hidden units required is unknown and can be impracticably large. By using multiple hidden layers, as shown in [66; 56], MLPs can partition the input space into exponentially more linear regions than a shallow network with the same number of neurons. For this reason they can more easily represent highly structured and complex functions. MLPs have obtained excellent results in pattern recognition tasks such as handwritten digit recognition [67], speech recognition [68].

Convolutional Neural Network CNNs are FNNs composed of convolutional layers. In a convolutional layer the input—typically two dimensional—is convolved with a set of learned kernels. The result of the convolution is then passed through a non-linear activation function, typically a ReLU. Other hidden layers often found in CNNs are pooling layers. Pooling layers down-sample the output of the convolutional layers (typically with the mean or maximum value for a region), therefore reducing overfitting and providing some translation invariance. After several convolutional and pooling layers, the output is a vector whose values are interpreted as the conditional probabilities of the input to belong to each class. A depiction of a CNN is presented in Figure 3.4.

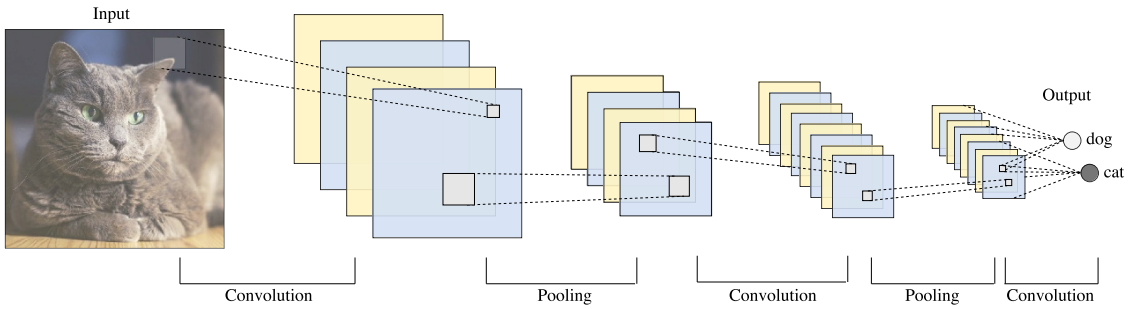

Figure 3.4: An example of a convolutional neural network in image classification. Figure adapted from [69]

CNNs have revolutionized the field of computer vision, leading to state-of-the-art results on complex image classification tasks such as the 1000 classes ImageNet Large-Scale Visual Recognition Challenge [53; 70].

In a FNN all samples are processed independently of each other. Due to the lack of context information FNNs cannot directly process sequential inputs such as audio, video and text. A fixed-size (causal or non-causal) window, concatenating the current feature vector with previous (and eventually future) feature vectors, is often used to provide context to the input. This approach however presents substantial

shortcomings, such as increased dimensionality—imposing the need for more data, longer training time and larger models—, unknown optimal window size, and fixed context available. RNNs, described in the following section, are more suitable to represent sequential inputs.

3.3 Recurrent Neural Networks

Introducing feedback connections in a neural network can provide it with past context information. This network architecture is known as recurrent neural network (RNN). As with FNNs, there are several types of RNNs introduced over the years, such as Elman networks [71], Jordan networks [72], time delay neural networks [73], long short-term memory [74] and gated recurrent units [75] networks.

For a sequence of input vectors $\{\mathbf{x}_1, ..., \mathbf{x}_T\}$, a simple RNN computes a sequence of hidden activations $\{\mathbf{h}_1, ..., \mathbf{h}_T\}$ and output vectors $\{\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_T\}$ as

$$\mathbf{h}_t = \mathcal{F}(\mathbf{W}^{\mathrm{xh}}\mathbf{x}_t + \mathbf{W}^{\mathrm{hh}}\mathbf{h}_{t-1} + \mathbf{b}^{\mathrm{h}}) \tag{3.24}$$

$$\hat{\mathbf{y}}_t = \mathcal{G}(\mathbf{W}^{\mathrm{h}\hat{\mathrm{y}}}\mathbf{h}_t + \mathbf{b}^{\hat{\mathrm{y}}}) \tag{3.25}$$

for all timesteps $t = 1, ..., T$, where the matrices $\mathbf{W}^{\star\star}$ denote the weights connecting two layers, $\mathbf{b}^{\star}$ are bias terms, and $\mathcal{F}$ and $\mathcal{G}$ activation functions. For a deep RNN with several stacked hidden layers[3], each hidden layer receives the output of the previous hidden layer (Figure 3.5).

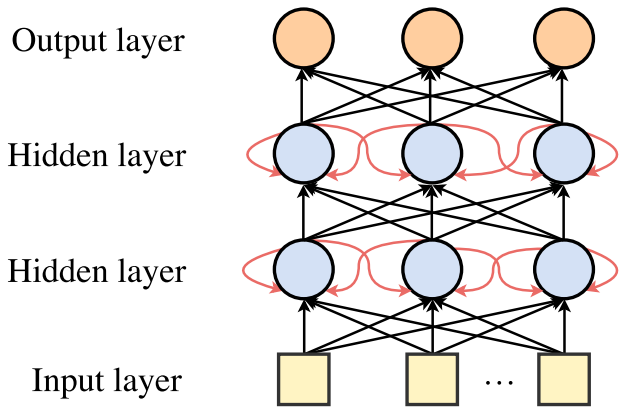

Figure 3.5: A recurrent neural network with 2 hidden layers, with feedback connections highlighted in red. Bias terms were omitted for clarity.

Despite the apparently minor modification, the effects are profound: in a RNN, information from previous time steps can in principle circulate indefinitely inside the network through the directed cycles, where the hidden layers also act as memory. These hidden activations create an *internal state*, represented by the $\mathbf{h}_t$ vectors for

[3]The concept of depth in RNNs however is not as well defined as in MLPs, since there are several ways to make an RNN deeper [76].

each hidden layer at a certain timestep, which is the result of the inputs history up to that time. Through the $\mathbf{h}_t$ vectors of the hidden layers the output of the network $\hat{\mathbf{y}}_t$ at time t becomes a function of all received inputs $\{\mathbf{x}_1, ..., \mathbf{x}_t\}$. RNNs therefore are very well suited to analyze sequential inputs. A simple RNN such as the one described here is also known as *vanilla RNN*.

RNNs, by the virtue of the internal memory, are far more expressive than FNNs. As stated in Section 3.2, a FNN can approximate any non-linear function on a compact domain to arbitrary precision. The equivalent result to MLP's universal approximation theorem is that a RNN with a sufficient amount of hidden units can approximate any dynamical system, *i.e.*, encode and execute any algorithm. Potentially a RNN is computationally as expressive as any Turing machine [77; 78].

A commonly used algorithm to train RNNs is *backpropagation through time* (BPTT) [79], a straightforward extension of the backpropagation algorithm. As shown in Figure 3.6, the idea behind BPTT is to unfold the network over time and then to apply the standard backpropagation algorithm as if it were a MLP. This unfolded representation shows that a RNN can be seen as a very deep FNN with a layer for each timestep and shared weights across time.

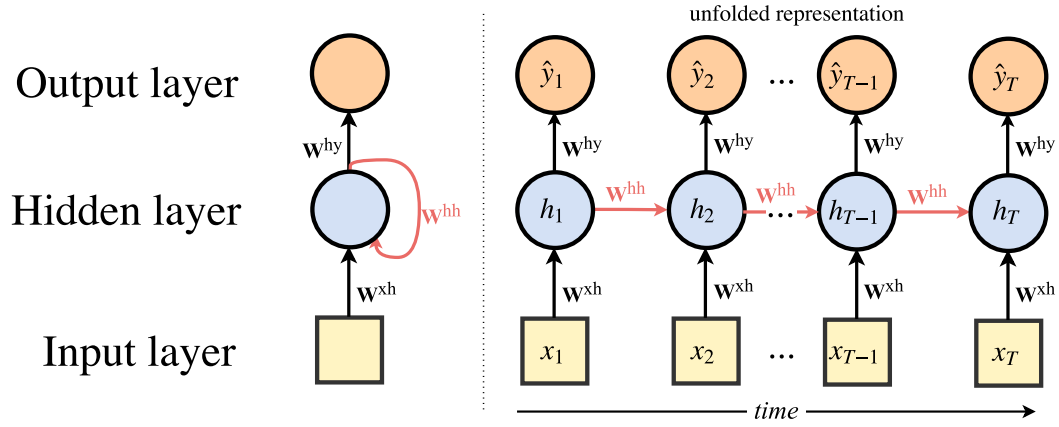

Figure 3.6: On the left, a recurrent neural network with 1 hidden layer and a single neuron. On the right, the same network unfolded in time over T steps. In this representation it is evident how an RNN can be interpreted as a deep MLP that has shared weight matrices. Bias terms are omitted for clarity.

Unfortunately, vanilla RNNs are difficult to train, and one of the main reasons is the phenomenon known as *vanishing gradient problem* [80; 81], which can be explained as follows. The cost function is calculated for every output $\hat{\mathbf{y}}_t$ and target $\mathbf{y}_t$ in the sequence, for t ranging from 1 to T, calling E_t the cost function computed at time t and $E = \sum_{t=1}^{T} E_t$ the cost function for the entire sequence. The difference from backpropagation in MLPs can be seen in the calculation of the gradients for the cost E_t with respect to the weight matrix $\mathbf{W}^{hh}$ at each step. As shown in the unfolded representation from Figure 3.6, this requires to propagate the gradients

several times through the same matrix $\mathbf{W}^{\text{hh}}$, since the output $\hat{\mathbf{y}}_t$ depends on the hidden state $\mathbf{h}_t$, which in turn depends on all the previous hidden states $\mathbf{h}_{t-1}$, $\mathbf{h}_{t-2}$, ..., $\mathbf{h}_1$. In the following remarks, we assume a different but equivalent formulation[4] of $\mathbf{h}$, that is $\mathbf{h}_t = \mathbf{W}^{\text{xh}}\mathcal{F}(\mathbf{x}_t) + \mathbf{W}^{\text{hh}}\mathbf{h}_{t-1} + \mathbf{b}^{\text{h}}$.

The expression of $\frac{\partial E_t}{\partial \mathbf{W}^{\text{hh}}}$ becomes, using the chain rule,

$$\frac{\partial E_t}{\partial \mathbf{W}^{\text{hh}}} = \sum_{m=1}^{t} \frac{\partial E_t}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_m} \frac{\partial \mathbf{h}_m}{\partial \mathbf{W}^{\text{hh}}}. \tag{3.26}$$

The term $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_m}$ can be further expanded using the chain rule, and it results in

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_m} = \prod_{j=m+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} = \prod_{j=m+1}^{t} \mathbf{W}^{\text{hh}\top} \text{diag}(\mathcal{F}'(\mathbf{h}_{j-1})), \tag{3.27}$$

where $\text{diag}(\mathbf{h})$ is a matrix whose diagonal elements are the components of $\mathbf{h}$, and has zeros elsewhere. This means that propagating the gradients from time t to time m will require the multiplication of $t-m$ matrices. More specifically, looking at the norms of the gradients, we obtain

$$\left\| \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_m} \right\|_2 = \left\| \prod_{j=m+1}^{t} \frac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}} \right\|_2 \leq (\beta_{\mathbf{W}} \beta_{\mathbf{h}})^{t-m} \tag{3.28}$$

where $\beta_{\mathbf{W}}$ and $\beta_{\mathbf{h}}$ are the upper bounds of $\left\|\mathbf{W}^{\text{hh}\top}\right\|_2$ and $\|\text{diag}(\mathcal{F}'(\mathbf{h}_{j-1}))\|_2$ respectively. In [81; 82] it was shown that if these upper bounds are smaller than 1, their product will vanish exponentially fast to 0, hence the name *vanishing gradient problem*. On the contrary, if these quantities are larger than 1, their product will explode to infinity, which leads to a phenomenon called *exploding gradient problem*.

The exploding gradient problem can be substantially attenuated by clipping the gradients whose norms are diverging [83], while the vanishing problem is more difficult to solve, making long term dependencies difficult to learn. Several techniques have been presented to overcome the difficulties of training RNNs, such as training with second order optimization methods (*e.g.* Hessian-free [84]), initialization of the recurrent weights with scaled identity matrix (IRNN) [85], unsupervised pre-training using restricted Boltzmann machines (RBMs) [86], linear autoencoder network initialization [87], and more complex architectures such as the long short-term memory.

3.4 Long Short-Term Memory

The long short-term memory (LSTM) architecture was proposed in 1997 by Hochreiter and Schmidhuber [74] as a solution to strongly mitigate the vanishing gradient

[4]For the details see [82]

problem. This architecture extends the vanilla RNN by replacing the simple neurons that have static self-connections with units called *LSTM memory blocks* (or *memory blocks*).

LSTM memory block An LSTM memory block is a subnet that contains one self-connected *memory cell* with its *tanh* input and output activation functions and three *gating neurons*—input, forget and output—with their corresponding multiplicative units. Figure 3.7 provides a representation of a memory block.

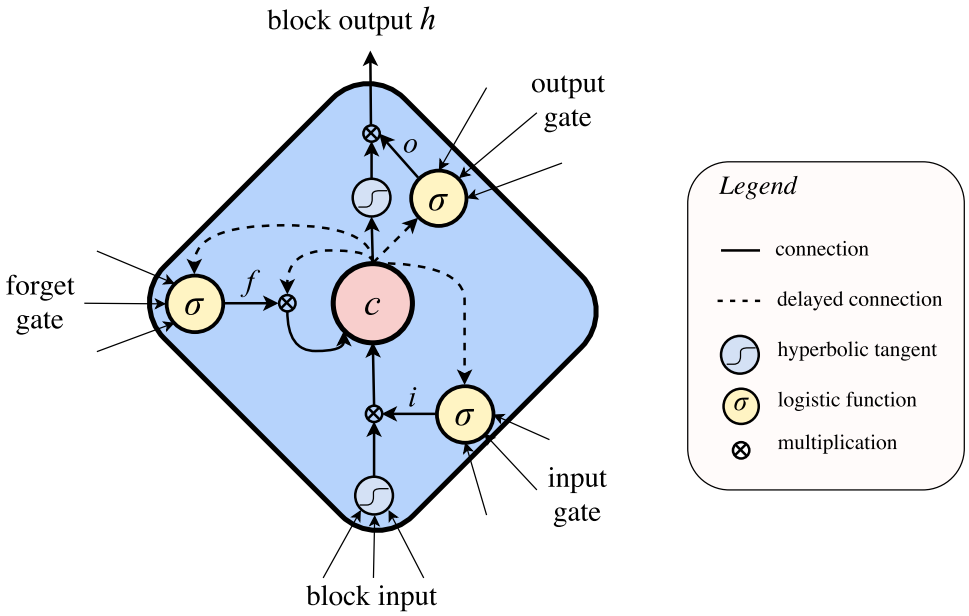

Figure 3.7: An LSTM block. The cell c has a recurrent connection with a fixed weight of 1 and a linear activation function. The three gating neurons are visible in yellow. Figure adapted from [88].

By analogy, the memory cell c can be compared to a computer memory chip, and the input i, forget f and output o gating neurons represent respectively *write*, *reset* and *read* operations. All gating neurons represent binary switches but use smooth sigmoidal activation functions (typically the logistic function, thus outputting in the range $[0, 1]$) to preserve differentiability. [88, p.33]

At timestep t the unit receives inputs at each of the four terminals (the input and the three gates). One part of the input derives from the current input vector $\mathbf{x}_t$ (or more in general, in case of a deep network, from the lower layer), the other from the hidden states at the previous timestep $\mathbf{h}_{t-1}$ of all memory blocks in the same layer. In addition to this, the value previously stored in the cell, *i.e.*, c_{t-1}, is also fed to the gating neurons through links called *peephole* connections. The inputs to the input terminal are summed and the result is passed through the *tanh* activation function. This activation is scaled by the activation of the input gate i_t and stored in the cell

state c_t summed to the previous cell state c_{t-1} scaled by the forget gate activation f_t. The final output of the memory block, which corresponds to the hidden state h_t, is computed as the product of the cell state c_t passed through the output activation function and the output gate activation o_t. Equation 3.24, defining the hidden activation $\mathbf{h}_t$, in an LSTM hidden layer is substituted by the set of equations:

$$\mathbf{i}_t = \sigma(\mathbf{W}^{\mathrm{xi}}\mathbf{x}_t + \mathbf{W}^{\mathrm{hi}}\mathbf{h}_{t-1} + \mathbf{W}^{\mathrm{ci}}\mathbf{c}_{t-1} + \mathbf{b}^{\mathrm{i}}) \tag{3.29}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}^{\mathrm{xf}}\mathbf{x}_t + \mathbf{W}^{\mathrm{hf}}\mathbf{h}_{t-1} + \mathbf{W}^{\mathrm{cf}}\mathbf{c}_{t-1} + \mathbf{b}^{\mathrm{f}}) \tag{3.30}$$

$$\mathbf{c}_t = \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \tanh(\mathbf{W}^{\mathrm{xc}}\mathbf{x}_t + \mathbf{W}^{\mathrm{hc}}\mathbf{h}_{t-1} + \mathbf{b}^{\mathrm{c}}) \tag{3.31}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}^{\mathrm{xo}}\mathbf{x}_t + \mathbf{W}^{\mathrm{ho}}\mathbf{h}_{t-1} + \mathbf{W}^{\mathrm{co}}\mathbf{c}_t + \mathbf{b}^{\mathrm{o}}) \tag{3.32}$$

$$\mathbf{h}_t = \mathbf{o}_t \tanh(\mathbf{c}_t) \tag{3.33}$$

where $\mathbf{c}_t, \mathbf{i}_t, \mathbf{f}_t$ and $\mathbf{o}_t$ are respectively the memory cell, input gate, forget gate and output gate activations of all the memory blocks in a layer, σ is the logistic function, $\mathbf{W}^{\star\star}$ are the weight matrices, and $\mathbf{b}^{\star}$ are bias terms. The peephole connections do not connect elements in different memory blocks, *i.e.*, the weight matrices $\mathbf{W}^{\mathrm{c}\star}$ from the cell to gate vectors are diagonal.

Gers et al. extended in [89] the original architecture proposed in [74] by adding the forget gate and the peephole connections. The forget gate in particular was later proven to be a fundamental component [90], and this extended version of LSTM is the most broadly used nowadays and also the one used in this work.

This complex architecture solves the vanishing gradient problem thanks to the recurrent connection of the cell c that has a weight of 1—which prevents the signal or the gradients from being scaled exponentially many times—and the cell's linear activation function that has unitary derivative. This is called the constant error carousel [74].

Bidirectional RNN and LSTM: In many tasks involving classification over temporally or spatially sequential inputs, we might be interested in looking also at future samples before outputting a label for the current timestep. For example, when classifying a handwritten letter inside a word it is beneficial to know the following letters as well as the preceding ones. Using a time-window to incorporate future samples, as explained in Section 3.2, poses strong limitations due, for example, to the fixed length of lookahead. Delaying the output by n timesteps would still impose a fixed lookahead and it would force the network to remember the original input—appeared n steps before—and the previous context throughout the delay. [88]

Bidirectional recurrent neural networks (BRNNs), introduced in [91], are a clever solution to the problem. In a BRNN each hidden layer is split into two separate layers, one reads the training sequences forwards ($\{\mathbf{x}_1, ..., \mathbf{x}_T\}$) and the other one

backwards ($\{\mathbf{x}_T, ..., \mathbf{x}_1\}$). The sequence is fully read and the hidden activations are stored for all timesteps in each of the two distinct layers. Finally, the computed activations are fed to the next layer, giving the network full and symmetrical context for both past and future samples of the input sequence. By substituting simple recurrent neurons in a BRNN with LSTM units we obtain a bidirectional long short-term memory (BLSTM) network, introduced in [92]. Figure 3.8 depicts a BRNN.

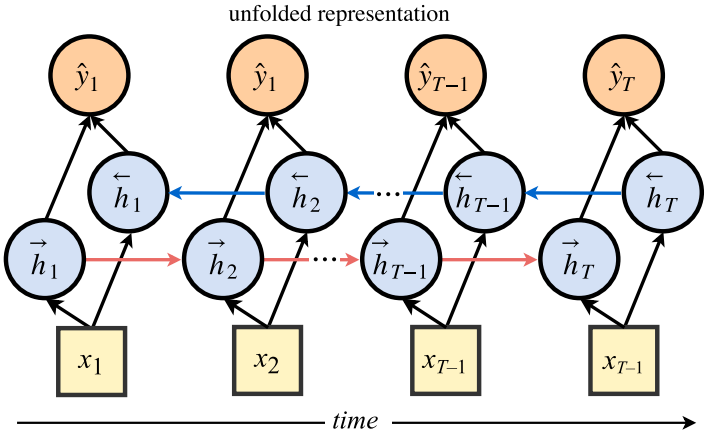

Figure 3.8: A bidirectional recurrent neural network with one hidden layer and two hidden neurons unfolded in time. Bias terms are omitted for clarity.

The violation of causality is not a concern in tasks where at test time the input sequence is completely available. This is the case of spatially sequential inputs (such as images), or temporally sequential inputs when the data was fully acquired before. Even data from a short amount of time ahead is sufficient in many tasks that need the output at the end of an input portion, such as in automatic speech recognition, where outputs are only required at the end of a word.

RNNs using LSTM and BLSTM units have advanced state-of-the-art results in many challenging tasks such as speech recognition [5], machine translation [93], unconstrained handwriting transcription [94] and generation [95], image captioning [96], language modeling [97], music modeling [98].

3.5 Data augmentation

As explained in Section 2.1, a model trained using supervised learning on a set of training examples should generalize to correctly classify new unseen data, avoiding overfitting to the training set. A challenging classification task requires a powerful model to be solved. A small neural network will not overfit easily to the training data, due to its limited flexibility, but will fail to fully describe the problem, thus yielding sub-optimal results. On the contrary, a large and deep neural network, with hundreds of thousands or millions of parameters, has enough expressive power to

properly solve the problem, but will be also very prone to overfit to the training data when this is scarce, hence failing generalization to new data.

By training the model on a large amount of data, it becomes more difficult for it to overfit. For this reason a huge amount of labeled data is a key requirement to train a large and robust model. Unfortunately collecting labeled data can be very expensive: in most cases a human expert has to manually go through all the data and assign a label. For datasets that contain millions or billions of samples this can become unfeasible. Data augmentation is an artifice used to obtain more labeled data without the need to collect it: the idea is to produce new data by manipulating and transforming the original data while preserving the labels, introducing variations that could be present in natural data.

A simple example of data augmentation in an image recognition task, is to horizontally mirror each of the training images and to assign to the new pictures the same labels of the original images, effectively doubling the size of the training set as depicted in Figure 3.9.

Figure 3.9: Two examples of augmenting an image dataset by flipping each picture along its horizontal axis. In both cases, a cat and a tree picture, the original labels correctly apply to the new images.

When creating new data by modifying the original dataset we have to make strong assumptions, *i.e.*, that the modifications made to the data do not interfere with the correctness of the labels assigned. The risk is to create data that is flawed or wrongly labeled, which might lead the model to make incorrect inferences. In the previous example from Figure 3.9, the underlying assumption was that the content of a picture of cat or of a tree remains the same if the image is horizontally flipped, and therefore the same label applies to the mirrored version. However, if the original images in the task were labeled, for example, after the handwritten letters represented—Figure 3.10—not all the flipped versions would correctly retain their labels.

This example shows that data must be augmented cautiously, and in fact usually most of the transformations applied are small variations of the data examples that preserve the labels. For example adding Gaussian noise to the training samples is

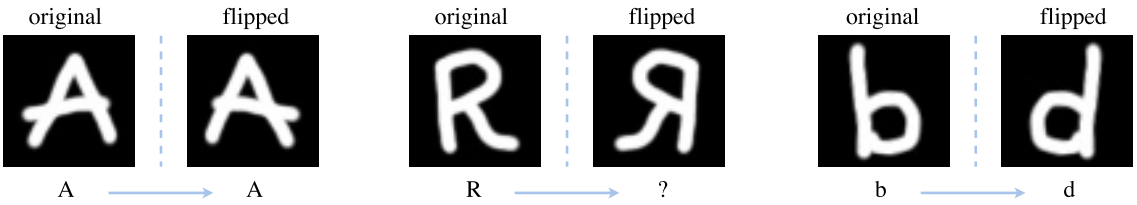

Figure 3.10: An example of a situation where the augmentation assumption is not always correct. While the letter 'A' would correctly retain its label when it is flipped, in the second case the image of the flipped letter 'R' would be labeled as 'R' despite not representing any letter. In the third case, the new image even changes its true label, turning from 'b' to 'd'.

a data augmentation technique widely used [99]: the assumption is that a small random variation in a sample's features does not change the label.

Several techniques of data augmentation are often used in image based tasks, using simple transformations of the images that preserve the labels, such as flipping, shifting, twirling and displacing. A model trained on an augmented dataset often achieves better generalization performance than if it is trained only on the original data [100; 101; 102]. In audio classification tasks some data augmentation techniques shown in the literature include vocal tract length perturbation, stochastic feature mapping, and elastic distortions [103; 104; 105] in ASR, and several pitch and time stretching techniques in [106] and [107] in music related tasks.

4. METHOD

This chapter describes our proposed method for a polyphonic sound event detection system using a multilabel BLSTM RNN. The system receives an audio recording as input and outputs the labels of the sounds recognized on every short time frame. The model is trained on time frames of a time-frequency representation of the mixture of sounds.

4.1 System overview

Here we present the overview of the system, briefly discussing all the steps in the pipeline, which correspond to the subsequent sections of this chapter. A depiction of the system is presented in Figure 4.1

The recorded audio signals are split into short frames and converted to a time-frequency domain representation. For each frame a certain class is marked as active if the corresponding sound is present, inactive otherwise. The data obtained is split into three non overlapping parts: training, validation and test set. Possibly, the training set is augmented to produce more data. The training and validation sets are fed to a multilabel RNN with several stacked BLSTM layers, which is trained to output the conditional probability that each class is active in every short frame. The training is halted when the validation error stops decreasing.

The system is then ready to be used on new, unseen data. New recordings are

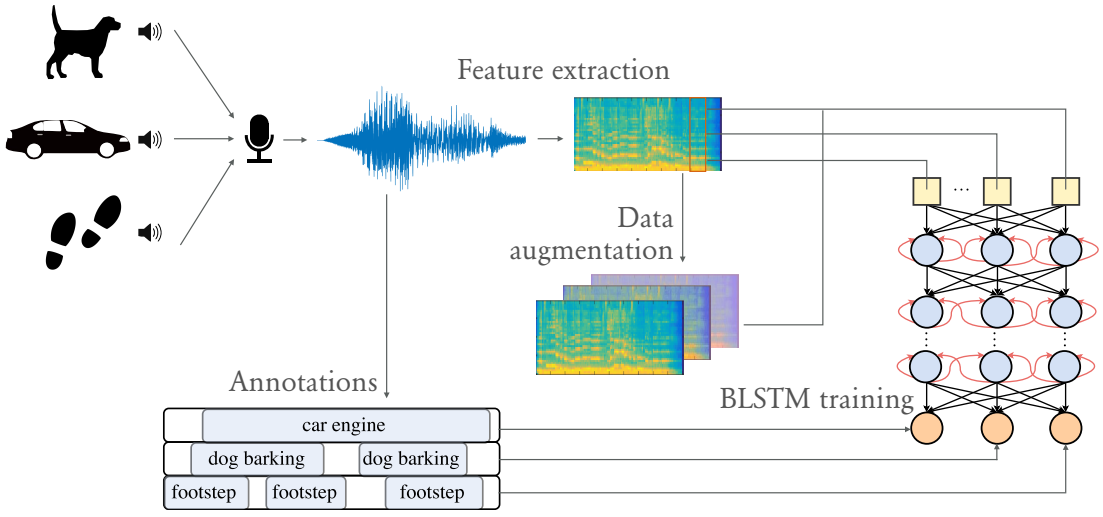

Figure 4.1: The system overview in the training phase.

processed in the same way and fed to the network. The generated outputs are interpreted as the probabilities that each class is active or not in a certain frame. The output probabilities are thresholded so that every class can be marked as either active or inactive.

To quantitatively evaluate the performance of the system we also test it on labeled unseen data, *e.g.*, the test set. We feed the test set to the network, threshold the output probabilities and compute several metrics to evaluate its performance. Experiments and results for the proposed approach are presented in Chapter 5.

4.2 Feature extraction

Figure 4.2 summarizes the steps performed in the feature extraction phase. The input to the system is a monaural or stereo audio recording. In the latter case, the signal is first converted to mono by averaging the two channels into a single one. Due to different recording conditions, the amplitude of the recorded signals can be quite different; in order to uniform them across recordings they are normalized to lie in $[-1, 1]$ as

$$\bar{\mathbf{s}} = \frac{\mathbf{s}}{\max(|\mathbf{s}|)}, \tag{4.1}$$

where $|\cdot|$ denotes the absolute value operator and is applied element-wise, the maximum operator computes the largest value of the vector, $\mathbf{s}$ is an original PCM audio signal, and $\bar{\mathbf{s}}$ its normalized version.

The normalized audio signal is then split into 50 milliseconds frames with 50% overlap. For each frame we compute its STFT and filter the magnitude spectrum into 40 mel energy bands, as described in Section 2.3. We then compute the logarithm of the mel energies obtained. The logarithmic energies—in addition to closer mimicking human perception of sound intensity—make the distribution of the data closer to Gaussian, a desirable feature in neural networks, as explained in Chapter 3. We then normalize each mel band by subtracting the mean value of each band over all recordings, and imposing unit variance (computing the constants on the training set), a standard procedure when working with neural networks.

All training data has to be labeled. To each feature vector $\mathbf{x}$ we associate a label

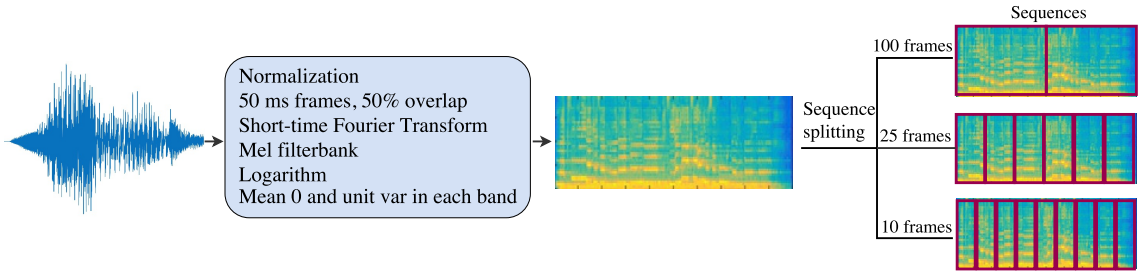

Figure 4.2: The pipeline of operations in the feature extraction phase.

vector $\mathbf{y} = (y_1, ..., y_K)$, where K is the number of classes and $y_k \in \{0, 1\}\ \forall k$. If the sound described by class k is not represented in $\mathbf{x}$ we set $y_k = 0$. Conversely, if class k is active we set $y_k = 1$. By horizontally concatenating all feature vectors $\mathbf{x}$ at different timesteps, we obtain the feature matrix $\mathbf{X}$, and similarly for the label vectors $\mathbf{y}$ we obtain the label matrix $\mathbf{Y}$.

For each recording we obtain a long sequence of feature vectors, which is then split into smaller sequences for training and testing. We split every original sequence at three different scales, *i.e.*, in non-overlapping length 10 sequences, length 25 sequences and length 100 sequences (corresponding to lengths of 0.25, 0.62 and 2.5 seconds respectively), as illustrated in Figure 4.3. This allows the network to identify patterns in the sequences at different timescales more easily. Using much longer sequences at training time would cause two main issues: the network might have difficulties with learning long time dependencies, and storing the hidden activations for all timesteps would impose impracticable memory requirements.

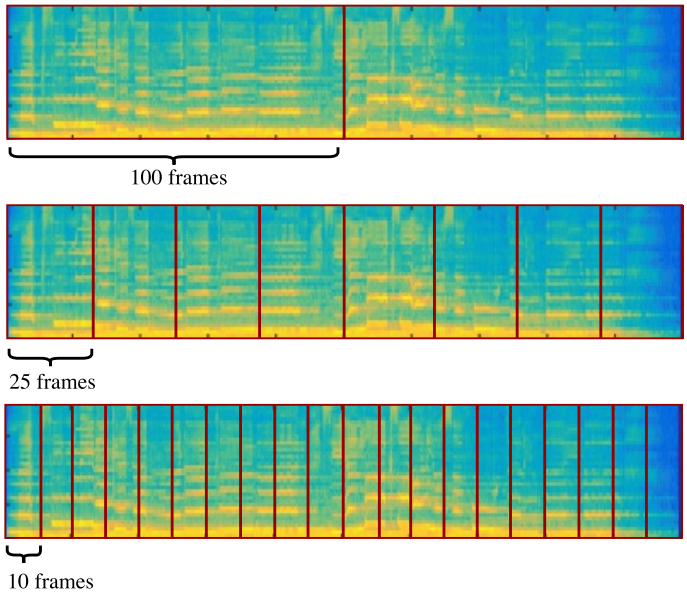

Figure 4.3: The sequences used for training are extracted at different lengths.

At test time we present the feature frames in sequences of 100 frames. On one hand having longer sequences would allow the model to access a richer context. On the other hand it would increase the delay for the predictions, reducing its real time suitableness. The maximum sequence length of 100 frames has been chosen as a compromise between good context information and fast predictions.

For the experiments presented in the next chapter, the mel features were extracted using the MIRToolbox [108], and then normalized and split into sequences in Matlab.

4.3 Proposed data augmentation

As explained in Section 3.5, data augmentation is a regularization technique used to reduce overfitting and to improve generalization. New labeled data is produced by manipulating the available training data while preserving the labels.

As an additional measure to reduce overfitting, which easily arises in case the dataset is small compared to the network size, we also augment the training set by simple transformations. All transformations are applied directly to the extracted features in frequency domain, making the transformations more efficient to compute and allowing to augment a dataset even when the recordings are not accessible, and only the extracted features are available. Similar techniques have been recently used in [106; 107], manipulating the signals in the time domain. All the data augmentation techniques presented in this section have been implemented in Matlab for the experiments reported in Chapter 5. It should be noted that these techniques are not exclusively tied to neural networks, and can be used with any other model.

Block mixing When two or more sound sources emit at the same time, the resulting mixture of sounds is the superposition of the individual waveforms. On this basis, new audio signals can be created by summing two (or more) different sections of the original recordings available. Since the content of both recordings is then present in the new signal, the resulting set of labels present in the new recording is the framewise union of the set of labels in the two original recordings.

As explained in Section 2.3, the linearity is also widely assumed to hold for the magnitude spectra, ignoring the phase information. Based on this assumption, it is then possible to directly sum the spectrograms extracted from two different sections to obtain the spectrogram of the mixture. The magnitude spectra should be summed before the logarithms are computed in the feature extraction phase. Alternatively, it is possible to approximate the exact log magnitude of the mixture using the mixture-maximization principle (*mixmax*) [109], according to which the interaction in the log-spectral domain between two signals S_1, S_2 can be explicited as follows:

$$\log(|S_1|+|S_2|) \approx \max(\log|S_1|, \log|S_2|). \tag{4.2}$$

A proof of optimality in a statistical framework for the mixmax principle was presented in [110].

On these bases, we can directly compute the log mel spectrograms for the combination of two sections from the individual features previously extracted (Figure 4.4). An example showing the small difference between a log mel spectrogram computed using these approximations and one extracted from the superposition of the waveforms is presented in Figure 4.5.

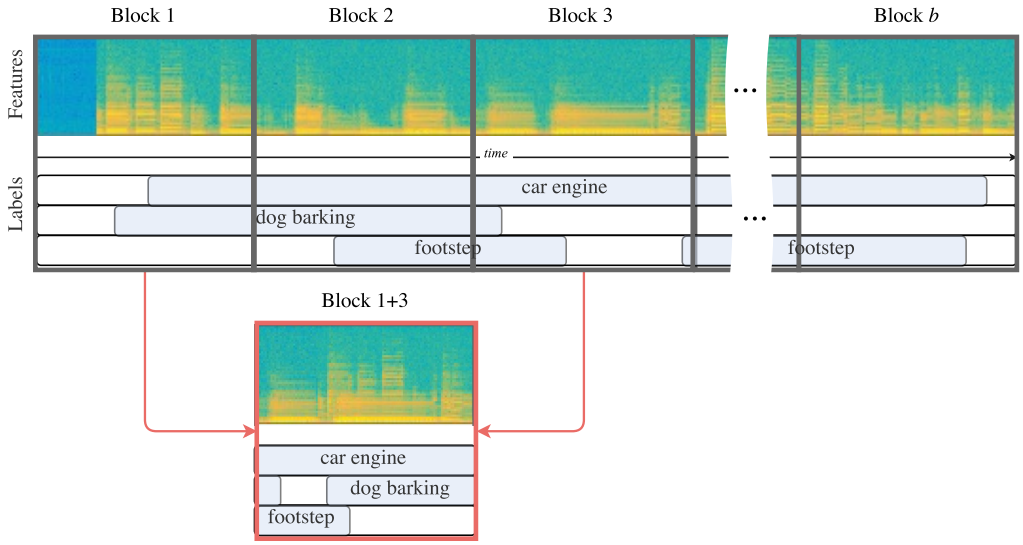

Figure 4.4: An example of the block mixing augmentation technique using 2 random blocks. The same operation is performed for every possible combination of p blocks at the time.

Formally, the feature matrix $\mathbf{X}$ is split into b non-overlapping and equally wide matrices (here called *blocks*), *e.g.*, $\mathbf{X}_1 = \{\mathbf{x}_1, ..., \mathbf{x}_t\}$, $\mathbf{X}_2 = \{\mathbf{x}_{t+1}, ..., \mathbf{x}_{2t}\}$ ... $\mathbf{X}_b = \{\mathbf{x}_{T-t+1}, ..., \mathbf{x}_T\}$. The label matrix $\mathbf{Y}$ is split accordingly in the same way. If b is not a factor of T, we first truncate $\mathbf{X}$ by discarding the last T (mod b) frames. All the $\binom{b}{p}$ combinations without repetitions of p blocks at the time are computed, where $p, b \in \mathbb{N}$ and $2 \leq p \leq b$. The combination of two blocks' feature matrices (*i.e.*, when $p = 2$) $\mathbf{X}_{j_1}$ and $\mathbf{X}_{j_2}$, and their label matrices $\mathbf{Y}_{j_1}$ and $\mathbf{Y}_{j_2}$ is defined as

$$\mathbf{X}_{j_1 j_2} = \max(\mathbf{X}_{j_1}, \mathbf{X}_{j_2}) \tag{4.3}$$

$$\mathbf{Y}_{j_1 j_2} = \max(\mathbf{Y}_{j_1}, \mathbf{Y}_{j_2}), \tag{4.4}$$

where max is applied element-by-element. For $p > 2$ the same computation is applied, computing the max over the p blocks to be combined.

A new block created using this process has a higher average polyphony than any of its addends alone. The polyphony level of the mixture of two (or more) original blocks can be seen as the sum of two (or more) independent discrete random variables. Therefore, the distribution of polyphony level in the new block is the discrete convolution of the distributions of polyphony level of the original blocks.

The main goal of this data augmentation technique is to produce high polyphony recordings that force the system to learn how to distinguish individual classes from complex combinations of sounds. The parameters b and p are chosen depending on the size and the distribution of polyphony level of the dataset. Increasing the value of b will produce a larger number of new blocks, but smaller in size. Increasing the value of p the average polyphony level grows linearly.

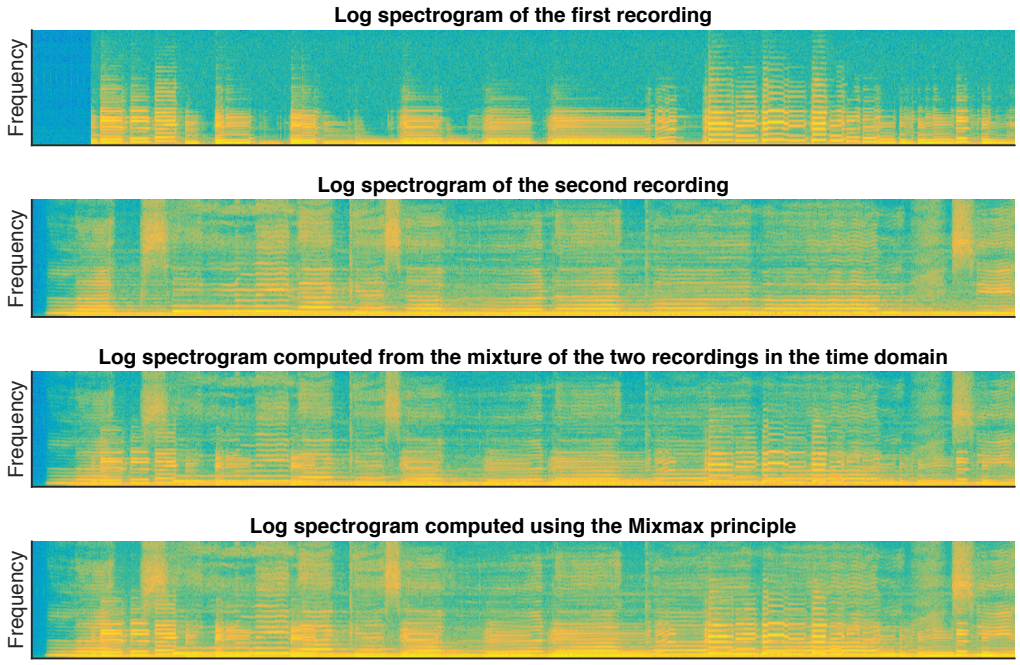

Figure 4.5: An illustration of the mixmax principle. The first two log spectrograms are computed from two different recordings; the third log spectrogram is computed from the mixture of the original recordings in the time domain; the fourth one is obtained directly from the first two, assuming the linearity of the magnitude spectra and applying the mixmax principle, *i.e.*, computing the pointwise maximum function of the two. It is evident from the figure that the approximation closely resembles the exact log spectrogram. This approximation is faster to calculate and only requires the original features already computed.

When the dataset contains recordings from real-life environments in different contexts (*e.g.* at the beach, in the office, at the restaurant, etc.), mixing two blocks from two different contexts might create very unlikely combinations of classes. In order to create more plausible data, we only mix blocks that belong to the same context, hence treating the recordings of each context independently from the others.

Time stretching A sound can be slightly slowed down or sped up while its source remains recognizable. Based on this assumption, we can generate more data by stretching the original recordings to make them slightly shorter or longer. The same stretching can be applied to the label matrices, in order to maintain consistency with the acoustic signals.

Simple dilation or compression of the waveform would affect both the pitch and the speed. Time stretching techniques typically work in the time-frequency domain, where time can be stretched independently from frequency [111]. For this reason we stretch the extracted mel spectrogram directly, instead of changing the speed of the original signals first and then extracting its features a second time. This is illustrated in Figure 4.6.

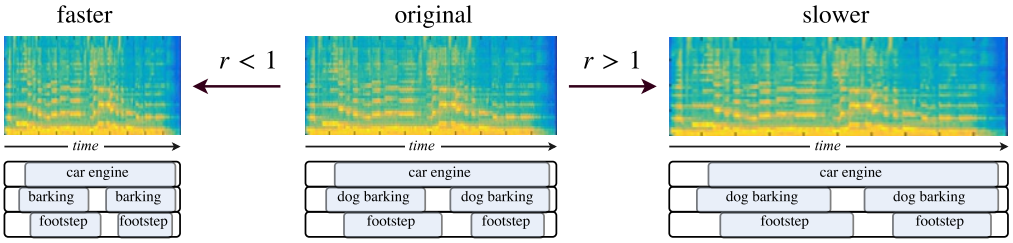

Figure 4.6: The time stretching augmentation. By stretching the time-frequency representation of a recording, it is possible to approximate the magnitude spectrum of a slowed down or sped up version of the same recording. The same stretch needs to be applied to the label matrices to maintain time coherence. The stretching factor r controls the amount of dilation or compression.

By treating the mel spectrogram and label matrices as images, it is possible to apply fast image stretching techniques. We horizontally stretch the mel spectrograms using linear interpolation by factors slightly smaller or bigger than 1. The new amount of frames is $r \cdot T$, where $r \in \mathbb{R}$ is the stretching factor and T the number of frames in the original signal. Thus, a stretching factor $0 < r < 1$ speeds up the recording and $r > 1$ slows it down. A new feature frame $\mathbf{x}$ between frames $\mathbf{x_t}$ and $\mathbf{x}_{t+1}$ is computed as

$$\mathbf{x} = \alpha \mathbf{x}_t + (1 - \alpha)\mathbf{x}_{t+1}, \tag{4.5}$$

where $\alpha \in [0, 1]$ linearly weights the contribution of the two original feature vectors and depends on the position of $\mathbf{x}$ between $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$.[1]

When the same stretching is applied to the label matrix $\mathbf{Y}$—in order to preserve the time alignment with the features—a choice has to be made on how to treat the new values which might be non-binary. For example, if a certain class k is active in $\mathbf{y}_t$ and inactive in the following label vector $\mathbf{y}_{t+1}$, when computing an interpolated $\mathbf{y}$ between the two, the resulting k-component of $\mathbf{y}$ is

$$y_k = \alpha 1 + (1 - \alpha)0 = \alpha \notin \{0, 1\}. \tag{4.6}$$

Not having exclusively binary labels does not represent a problem with the set up used in this work. If for different algorithms the labels need to be strictly binary, it is possible for example to round the components to the nearest integer.

Sub-frame time shifting The spectral features computed for a given audio signal depend on the alignment between the signal and the STFT windows. By shifting the windows in time by a few milliseconds the resulting spectrogram will be slightly different than the original one. We mimic small time shifts of the windows—at

[1]For the open form of α see for example *Linear Interpolation* on Springer Encyclopedia of Mathematics.

sub-frame scale—by linearly interpolating new feature frames in-between existing frames, thus retaining the same frame rate. Figure 4.7 illustrates the procedure.

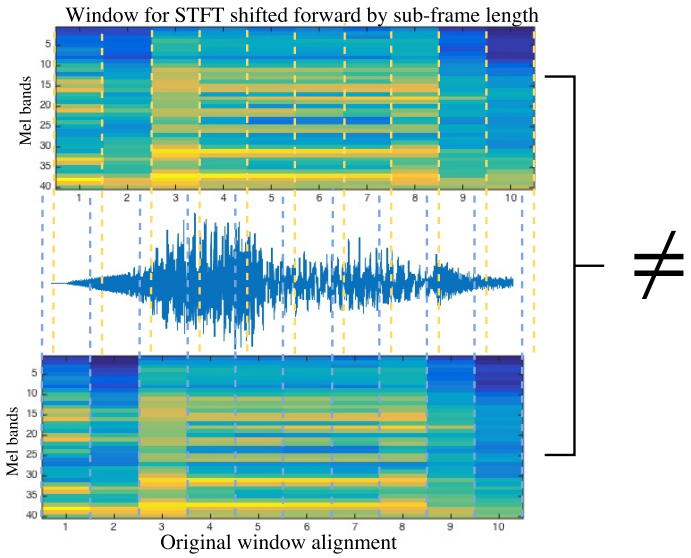

Figure 4.7: The sub-frame time shifting augmentation, omitting for clarity every other frame due to the 50% window overlap in the STFT. By shifting the windows used to compute the STFT in time by a small amount—*i.e.*, less than half the window size—the spectral representation computed is slightly different than the original one. Therefore, the new frames can be used as new data.

Formally, we apply the same algorithm used for time stretching, but this time using a constant value of α throughout the recordings and interpolating always one new frame in between every pair of subsequent frames. The process can be applied several times using different values of $\alpha \in (0, 1)$. The same procedure is repeated on the label matrix $\mathbf{Y}$ for time consistency. The aforementioned considerations on non-binary targets in between activity transitions apply in this case too.

4.4 Proposed neural network

The characteristics of the task suggest that deep neural networks are potentially an excellent tool for polyphonic SED. It is difficult to design discriminative high-level features by hand, there is potential to collect a large amount of data, and there are large variations in the way the samples from each category might appear. While the aforementioned issues are shared by monophonic SED as well, there is one more key element distinctive of polyphonic SED, that is computation. The SED system should be capable of recognizing unseen combinations of sounds, and learn to recognize individual events even if they only appear in combinations with others in the training data. Moreover, as mentioned in Section 2.4, there is evidence that deep neural networks have achieved excellent results on the same task, such as the FNN in [11]. We approach the problem of polyphonic SED using a multilabel

BLSTM RNNs with multiple hidden layers, to map the acoustic features to class activity indicator vectors. By using RNNs we hope to overcome the limitations of FNNs in dealing with short context information.

As it has been explained in Section 2.1, using a multilabel model allows in principle the system to discover and model correlations among classes, possibly improving its classification performance. Another advantage, compared to having one binary model for each class, is that there is only one model that needs to be trained and run at test time.

The input layer consists of 40 nodes, each reading one component of a log mel energies input frame. The input layer is fully connected to the first hidden layer, a recurrent BLSTM layer. In a BLSTM layer, as explained in Section 3.3, half of the memory blocks scan the input sequence forwards and half scan it backwards. For this reason, for a given sequence the activations for both halves need to be fully computed and stored before the outputs can be fed to the next layer. Each hidden layer is a BLSTM layer fully connected to the next hidden layer. A simplified representation of the BLSTM RNN is presented in Figure 4.8.

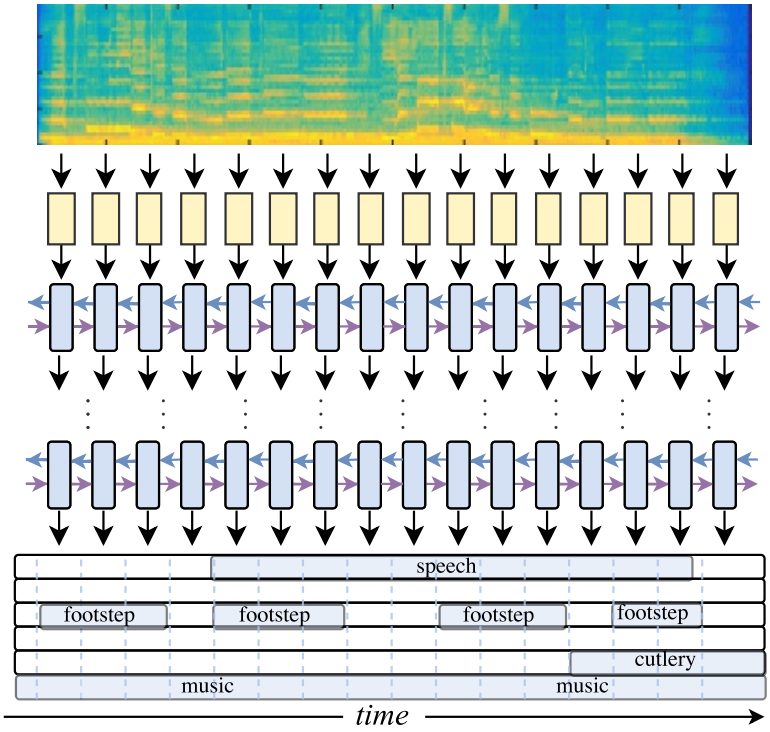

Figure 4.8: A depiction of the proposed BLSTM RNN unfolded over time, where each block represents a layer of neurons.

Since it is not possible to know the best network configuration for a certain dataset a priori, several networks need to be trained with different architectures, *i.e.*, varying the number of hidden layers and hidden units. The configuration that gives the best results on the validation set is then selected for testing.

The output layer has logistic activation functions and one neuron for each class. The output of the network at time t is a vector $\hat{\mathbf{y}}_t \in [0,1]^K$, where K is the number of classes. Its components $\hat{y}_k$ can be interpreted as probabilities that each class is active or inactive in frame $\mathbf{x}_t$. These outputs do not have to sum up to 1, since several classes might be active simultaneously. Thus, contrarily to most multiclass approaches with neural networks, the outputs are not normalized by computing the softmax function. Finally, the continuous outputs are thresholded to obtain binary indicators of class activities for each timestep. Contrarily to the FNN in [11], where the outputs are smoothed over time using a median filter on a 10-frame window, we do not apply any post-processing since the outputs from the RNN are already smooth.

For training, a fast gradient descent algorithm such as RMSprop can be used as an alternative to steepest descent with momentum for faster convergence. In order to further speed up the training, the networks can be trained on mini-batches of few hundreds sequences, updating the parameters after processing each mini-batch. This also makes the gradients less noisy than in online training, and allows parallel processing of multiple sequences at the time using GPU.

Injecting noise in the features is a common technique used in machine learning to improve the robustness of a model. In this work we corrupt the log mel energy features extracted by summing a noise vector $\boldsymbol{\varepsilon}$ whose components are values extracted from a Gaussian distribution, independently one to the others, with mean μ and standard deviation σ. While μ is typically set to 0, a good value of σ—which depends on the dataset and the network configuration—needs to be determined by testing several different ones. The amount of noise added to each feature vector is changed in every epoch of neural network training.

A wide range of experiments was conducted on the two datasets presented in Section 5.1.1 and 5.1.2, varying the parameters of the RNNs to maximize the performance on the validation sets. The final configurations used for both datasets are reported in the same sections.

5. EVALUATION

This chapter presents the sound event datasets used in this work and describes in detail the metrics used to evaluate the system's performance. We then present the results obtained and compare them to the state-of-the art approach from the literature.

5.1 Datasets

We evaluate the approach proposed in Chapter 4 on two sound event datasets, described in Section 5.1.1 and 5.1.2. For each dataset we present how the recordings were collected and labeled, what classes are present and how they are distributed. Furthermore we report the following statistics:

- *Label cardinality:* it corresponds to the average polyphony level. It is computed as the average number of classes active per sample, formally

$$\text{Label cardinality} = \frac{1}{N}\sum_{n=1}^{N}|\mathbf{y}_n|, \tag{5.1}$$

  where $|\mathbf{y}_n|$ is the number of non-zero elements in vector $\mathbf{y}_n$.

- *Label density:* it is the average number of classes active per sample divided by the total number of labels, computed as

$$\text{Label density} = \frac{1}{N}\sum_{n=1}^{N}\frac{|\mathbf{y}_n|}{K} \tag{5.2}$$

  where K is the cardinality of the label set.

5.1.1 Real life recordings dataset

This dataset, referred to as *real life recordings dataset* (RLRD) in this work, was provided by the Audio Research Group at Tampere University of Technology [112]. It consists of stereo recordings 10 to 30 minutes long, from ten real-life contexts, namely: basketball game, beach, inside a bus, inside a car, hallway, office, restaurant, shop, street and stadium with track and field events. Each context has 8 to 14 recordings, for a total of 1133 minutes distributed over 103 recordings. The

technical equipment used consisted of a binaural microphone (Soundman OKM II Klassik/Studio A3) worn by a walking human agent, and a digital recorder (Roland Edirol R-09). Recordings were acquired at 44.1 kHz sampling rate and 24-bit resolution. To ease the annotation process and increase its accuracy, also video recordings were acquired while recording audio.

The recordings were annotated manually, marking the beginning and ending moments of all clearly audible sound events. Short repetitive sounds, such as a basketball hitting the floor, were annotated as single long events; long quasi-continuous events, such as speech in a conversation, were annotated as multiple events when a pause in the conversation was clearly perceivable.

The sound events were annotated within 60 classes, including *speech, applause, music, brake squeak, keyboard*; plus 1 class for rare or unknown events marked as *unknown*, for a total of 61 classes. All the events appear multiple times in the recordings; some of them are present in different contexts, others are context-specific. All 61 classes are reported in Table 5.1.

The average polyphony level for this dataset is 2.53, for a label density of 0.041. The distribution of polyphony levels across all recordings is illustrated in Figure 5.1.

Table 5.1: The list of classes present in the RLRD.

No.	**Event**	**No.**	**Event**	**No.**	**Event**
1	*applause*	21	*click*	41	*refrigerator*
2	*background*	22	*coins keys*	42	*road*
3	*ball hitting floor*	23	*coughing*	43	*seatbelt*
4	*beep*	24	*crowd sigh*	44	*shoe squeaks*
5	*bicycle*	25	*crowd walla*	45	*shopping basket*
6	*bird*	26	*dish washer*	46	*shopping cart*
7	*brakes squeak*	27	*dishes*	47	*sigh*
8	*breathing noises*	28	*dog barking*	48	*signal horn*
9	*bus*	29	*door*	49	*sliding door*
10	*bus door*	30	*engine off*	50	*sneezing*
11	*car*	31	*footsteps*	51	*speech*
12	*car door*	32	*keyboard*	52	*traffic*
13	*car engine starts*	33	*laughter*	53	*turn signal noise*
14	*cash register*	34	*motor noise*	54	*water splashing*
15	*cat meaowing*	35	*motorbike*	55	*wheel noise*
16	*chair*	36	*mouse scrolling*	56	*whistling*
17	*cheering*	37	*music*	57	*wind on trees*
18	*child*	38	*paper movement*	58	*windscreen wipers*
19	*clapping*	39	*pressure release*	59	*wrapping*
20	*clearing throat*	40	*referee whistle*	60	*yelling*
61	*unknown*				

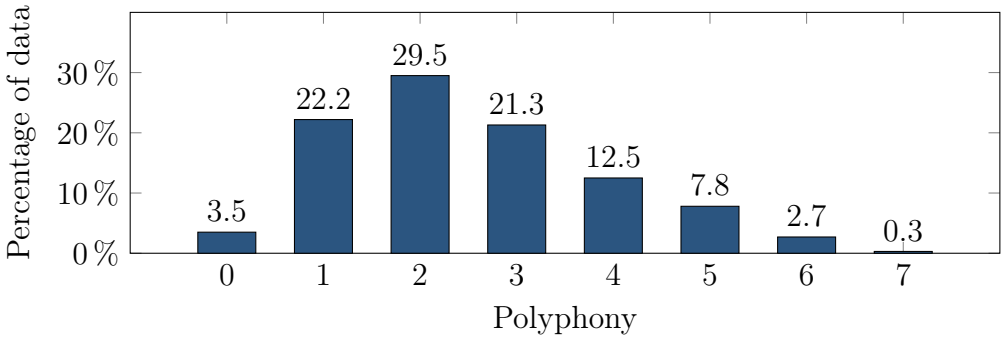

Figure 5.1: Distribution of polyphony level across the RLRD.

The dataset was split into training, validation and test set (about 60%, 20% and 20% of the data respectively) in a 5-fold manner. For every fold, each recording was only assigned to training, validation or test. All results are presented as averages of the 5-fold cross validation results, with the same train/validation/test partitions used in [11] on the same dataset. Sound event detection techniques tested on this dataset include a work using GMM-HMM [13], a purely NMF based approach [15], and the deep FNN from [11] that will serve as a baseline.

At the time of writing this thesis, this is still one of the few large and annotated polyphonic real-life environment sound event dataset in the world. Due to the vastness of the dataset and the intrinsic difficulty in manually annotating multiple concurrent sound events, the annotations are not always accurate. This might have an effect on the classification performances, for example putting a limit to the maximum accuracy that can be reached.

Amount of data augmentation Concerning the data augmentation techniques described in Section 4.3, we expand this dataset by approximately 16 times. A 4-fold increase comes from the time stretching (using stretching coefficients of 0.7, 0.85, 1.2, 1.5), 3-fold increase from sub-frame time shifting and 9.5-fold increase from block mixing (mixing 2 blocks at the time, using 20 non-overlapping blocks of equal size for each context). We do not test other amounts or parameters of augmentations.

5.1.2 Artificial mixtures dataset

This dataset, referred to as AMD (artificial mixtures dataset) in this work, was also provided by the Audio Research Group. Contrarily to RLRD, which is composed of recordings from real-life scenarios, this dataset is an artificial mixture of studio-recorded sounds from the *BBC Sound Effects Library*.

The original BBC dataset consists of in-studio stereo recordings. Each file contains a single sound event, whose class is marked on the file name. From about a

hundred classes available, 63 sound classes were selected and are listed in Table 5.2. In this dataset as well, several recordings are available for each class.

Each file contains a few milliseconds of silence at the beginning, at the end, and in some cases—such as intermittent sound events—in the middle of the recording. In order to avoid marking these silent frames as active, which might mislead the system during training, an automatic labeling procedure was applied. For each recording the mean energy of the signal was first computed, and only those frames whose energy was greater than 0.15 times the mean energy of the whole recording were marked as active.

To obtain a polyphonic and continuous recording for training, the labeled audio files were randomly extracted, then assigned a random position in time, and finally overlapped by summing their waveforms. To produce a validation and test set the same procedure was applied, using different files for each set. The resulting labeled recordings for training, validation and test set are 1940, 240 and 240 minutes long respectively. The average polyphony level for this dataset is 1.5, with a label density of 0.024. The distribution of polyphony levels across the three long recordings is reported in Figure 5.2. Finally, the audio features were again extracted as explained

Table 5.2: The list of classes present in the AMD.

No.	**Event**	**No.**	**Event**	**No.**	**Event**
1	*bird singing*	22	*computer typing*	43	*cutting knife*
2	*horsewalk*	23	*paper tear*	44	*mixer*
3	*dog barking*	24	*paper movement*	45	*glass smash*
4	*cattle*	25	*pen writing*	46	*shopping cart*
5	*cat meowing*	26	*photocopier*	47	*plastic bag*
6	*goat*	27	*human laughing*	48	*coins*
7	*wind*	28	*baby laughing*	49	*rain*
8	*elevator door*	29	*baby crying*	50	*alarms&sirens*
9	*vending machine*	30	*human eating*	51	*chainsaw*
10	*sliding door*	31	*human coughing*	52	*gun shot*
11	*car*	32	*door bell*	53	*police siren*
12	*yacht*	33	*alarm clock*	54	*Big Ben clock*
13	*land rover*	34	*washing machine*	55	*footsteps*
14	*car engine start*	35	*hair dryer*	56	*lawn mower*
15	*car turn signal*	36	*camera shutter*	57	*thunder*
16	*windscreen wipers*	37	*toilet*	58	*crowd cheering*
17	*bus*	38	*fan*	59	*crowd conversation*
18	*bicycle*	39	*broom*	60	*crowd yelling*
19	*car door*	40	*water sink*	61	*crowd applause*
20	*motorcycle*	41	*vacuum*	62	*crowd whistling*
21	*car seat belt*	42	*fireplace*	63	*unknown*

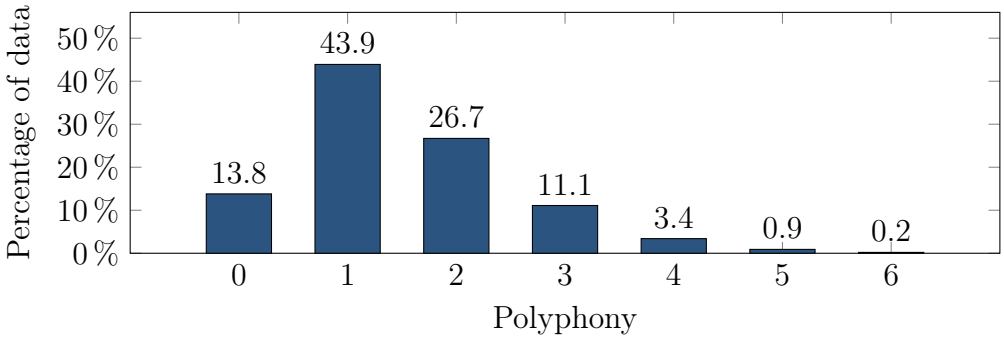

Figure 5.2: Distribution of polyphony level across the AMD.

in Section 4.2.

High polyphony segments in this dataset are more difficult to classify correctly compared to the RLRD. In the latter, only a small number of combinations are likely to appear, more specifically the combinations of the subset of events present in any context. Therefore, the class combinations present in the high polyphony segments of RLRD are likely to appear both in the training and test set; in principle the network might learn to detect the characteristic features of these combinations without recognizing the individual classes that are present. On the contrary, in this artificial dataset, because of the random process used to combine the events when the mixtures were generated, any combinations of events is possible. It is thus very unlikely that a combination of 4 or more labels in the test data has also appeared in the training data.

The advantage of this dataset compared to the RLRD is that the annotations are very precise, allowing to obtain a more accurate system performance estimation. The main disadvantage comes from the random extraction and overlapping of the files, which makes any combination of events possible and thus further from the real scenarios. For the same reason, there are no correlations among classes and no context information, contrarily to the RLRD.

This dataset was not previously used in other published works. For a more insightful evaluation of the proposed approach, we compare our results on this dataset to those of a FNN similar to the one used in the baseline [11]. To make the comparison with the proposed approach as fair as possible, both our RNN and the FNN configurations are the same used for the RLRD dataset. The only difference is in the output layer of both networks, that have 63 neurons instead of 61, in order to match the number of classes in this dataset.

Amount of data augmentation For this dataset we test the different data augmentation techniques separately. Time stretching is performed using the same stretching coefficients used in RLRD, *i.e.*, 0.7, 0.85, 1.2, 1.5, producing approxi-

mately 4 times more data. Sub-frame time shifting is applied 4 times, leading to a 4-fold increase. Block mixing, which allows to create more diverse data, is used three times with different parameters:

- 2 blocks at the time, using 20 non-overlapping blocks of equal size. This produces a 9.5-fold increase.
- 3 blocks at the time, using 7 non-overlapping blocks of equal size. This produces a 5-fold increase.
- 3 blocks at the time, using 4 non-overlapping blocks of equal size. This produces a 1-fold increase.

5.2 Evaluation procedure

As explained in Section 2.1, evaluating the performance of a multilabel classifier is not straightforward and there are several metrics that can be computed to evaluate different aspects of performance.

The goal is to assess how close the network prediction $\hat{\mathbf{y}}$ is to the target $\mathbf{y}$. For each class k there are 4 possible combinations of target and output $(y_k, \hat{y}_k)$, as shown in Figure 5.3:

- $(1, 1)$, *True positive (TP):* class k is correctly detected as active.
- $(0, 0)$, *True negative (TN):* class k is correctly detected as inactive.
- $(0, 1)$, *False positive (FP):* class k is incorrectly detected as active.
- $(1, 0)$, *False negative (FN):* class k is incorrectly detected as inactive.

From these definitions it is possible to define *precision* and *recall*:

$$\text{precision} = \frac{TP}{TP + FP} \tag{5.3}$$

$$\text{recall} = \frac{TP}{TP + FN}. \tag{5.4}$$

Neither precision nor recall alone provide an accurate picture of the performance. If a system were constantly silent for all frames, *i.e.*, outputting that all classes are inactive ($\hat{\mathbf{y}} = \mathbf{0}$), there would be no false positives, and thus precision = 1. If a system output all classes as active for all frames ($\hat{\mathbf{y}} = \mathbf{1}$) there would be no false negatives, and thus recall = 1. A well-performing system would maximize both measures simultaneously.

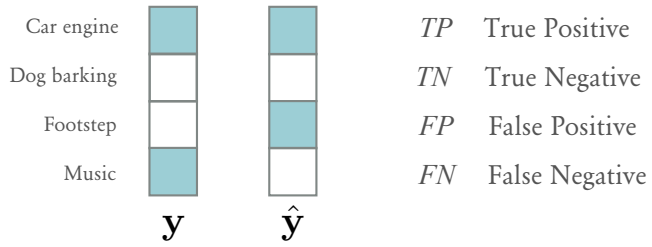

Figure 5.3: A depiction of a true positive, a true negative, a false positive and a false negative. A full light blue square corresponds to a class that is marked as active.

By computing the harmonic mean of precision and recall we can define the $F1$ score (from now on simply $F1$):

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{2TP}{2TP + FP + FN}. \tag{5.5}$$

$F1$ acts approximately as a smooth version of the minimum operator. Therefore good scores on both precision and recall will be favored over very good performance on one and poor on the other. This makes $F1$ a more meaningful and robust metric, widely used in the context of SED [11; 12; 13; 15; 113; 114].

There are several ways of extending the $F1$ for time series, here is a list of the ones used in this work:

- **Mean frame-wise F1:** $F1$ is computed independently for each frame and the results are averaged throughout all frames.

$$F1_{\text{AvgFram}} = \frac{1}{N} \sum_{j=n}^{N} F1(\mathbf{y}_n \hat{\mathbf{y}}_n). \tag{5.6}$$

  where N is the number of test observations. This metric assigns the same weight, *i.e.*, $\frac{1}{N}$, to frames with different polyphony levels.

- **1 second F1:** in SED tasks it is sometimes considered to be more important to detect that a certain event has happened than to correctly identify its exact beginning and ending moment. A metric can be defined such that any event active in the predictions or targets for at least one frame of a s seconds block is marked as active for the whole block. This allows for a small amount of tolerance and a coarser time resolution. In this work we used $s = 1$ (as proposed in [22]), therefore since the frames are 50 ms long with a 50% overlap, 40 concatenated frames represent a 1 second block. A block prediction and

target vector are computed as follows

$$\mathbf{y}^{\text{block}} = \max_{t-39 \leq j < t} (\mathbf{y}_j) \tag{5.7}$$

$$\hat{\mathbf{y}}^{\text{block}} = \max_{t-39 \leq j < t} (\hat{\mathbf{y}}_n) \tag{5.8}$$

where max is applied element by element, as it is shown in Figure 5.4. The block predictions and targets are computed on non-overlapping blocks. TP, FN and FP are then computed on $\mathbf{y}_{\text{block}}$ and $\hat{\mathbf{y}}_{\text{block}}$ for each block, the results are used to compute $F1$ and they are averaged over all blocks.

$$F1_{\text{1-sec}} = \frac{40}{N} \sum_{n=1}^{N/40} F1(\mathbf{y}_n^{\text{block}}, \hat{\mathbf{y}}_n^{\text{block}}). \tag{5.9}$$

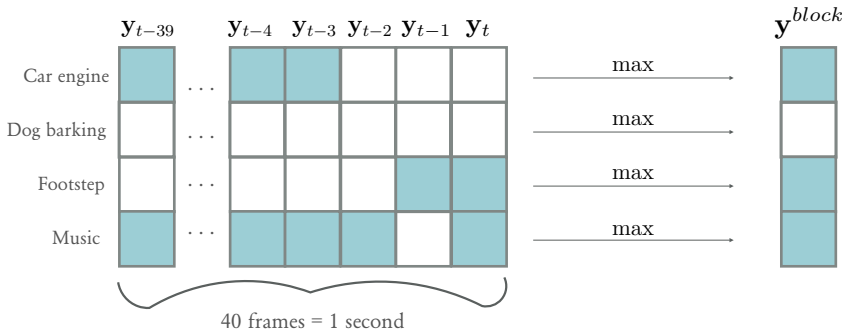

Figure 5.4: An example of the process used to compute the target vectors for a 1 second block. All classes that are active in at least one frame in the block are marked as active for the enitre block. The same approach is used to compute the predictions in 1 second blocks $\hat{\mathbf{y}}^{\text{block}}$.

- **Micro-average F1:** all TP are summed across all frames, and the same is done for FP and FN. Micro-average $F1$ ($F1_{\text{Micro}}$) is obtained applying equation 5.5 using the computed results.

$$F1_{\text{Micro}} = \frac{2 \sum_{n=1}^{N} TP}{2 \sum_{n=1}^{N} TP + \sum_{n=1}^{N} FP + \sum_{n=1}^{N} FN}. \tag{5.10}$$

  This metric assigns higher weights to frames with higher polyphony.

The three metrics are likely to display similar results. For simplicity and readability we focus the analysis on certain metrics when evaluating the performance.

Detecting sounds when multiple sources are active at the same time becomes generally more challenging as the polyphony level increases. In order to measure

performances across different polyphony levels we also compute $F1_{\text{AvgFram}}$ separately for each of them.

In the case of RLRD, as described in Section 5.1.1, the recordings were collected from 10 different contexts and for each context there are several recordings in the test set. In line with previous works on this dataset, we compute the scores individually for each context—referred to as *context scores*—by first concatenating the individual recordings of each context into one long recording. The overall scores for the dataset are then computed as the average of the 10 context scores. For the AMD dataset, which has no context information and a single long recording for test purposes, we only report the overall scores.

5.3 Neural networks experiments

The networks chosen based on validation results for both datasets, have an input layer with 40 units, each reading one component of the feature frames, 4 hidden layers with 200 LSTM units each—100 reading the sequence forwards, 100 backwards—and one output neuron with logistic activation for each class. For both datasets we train one network with the original data only—which for RLRD is the same used in previous works—and one or more using the data augmentation techniques reported in Section 4.3 to further reduce overfitting. In order to compare the performance of BLSTM to unidirectional LSTM, we also train a similar network architecture without bidirectional units on the same datasets without augmentation.

The networks are initialised with uniformly distributed weights in $[-0.1, 0.1]$ and trained using RMSE as a cost function. Training is done by back propagation through time (BPTT) [79], as explained in Section 3.3. The extracted features are presented as sequences clipped from the original data—in sequences of 10, 25 and 100 frames—in randomly ordered mini-batches of 600 sequences, in order to allow parallel processing. Due to limited processing power, in our experiments the features computed from the augmented data are only presented as sequences of either 25 or 100 frames. After a mini-batch is processed, the weights are updated using RMSProp with a step rate $\eta = 0.005$ and decay term $\gamma = 0.9$. If the training cost starts to rise during training, we lower the step rate η to 0.001. The training is halted if the validation cost does not decrease for 20 consecutive epochs. For RLRD we use Gaussian input noise with $\sigma = 0.2$ during training; for AMD, such a high input noise hurts the performances, so we set it to $\sigma = 0.03$. All the hyperparameters were chosen based on the validation sets, for RLRD only on the validation partition of the first fold. At test time we present the feature frames in sequences of 100 frames, and threshold the outputs with a fixed threshold of 0.5, *i.e.*, we mark an event k as active if $\hat{y}_k \geq 0.5$, inactive otherwise.

For each experiment in RLRD we train 5 networks with different random initial-

isations, *i.e.*, 5 networks per fold for a total of 25 networks. We then select in each fold the network that has the highest performance on the validation set and use it to compute the results on the test data. Since the training times for AMD are very long, requiring more than one week to train the networks on the augmented datasets, we only train a single network for most of the experiments with data augmentation.

For all the experiments presented in this chapter, the networks were trained on a Tesla K40t GPU with the CUDA/C++ open-source toolkit "Currennt" [115]. Since the only optimizer available in the package is stochastic gradient descent plus momentum (SGD+M)—which is often too slow to converge—we implemented RMSprop; this allowed us to train much larger models and to use more augmented data.

5.4 Results

5.4.1 Real life recordings dataset

In Table 5.3 we compare the average scores over all contexts for the FNN in [11] to our BLSTM and LSTM networks trained on the same data, and BLSTM network trained with the augmented data. The FNN uses the same features but at each timestep reads a concatenation of 5 input frames (the current frame and the two previous and two following frames). It has two hidden layers with 1600 hidden units each, downsampled to 800 with maxout activations.

Table 5.3: Overall $F1$ scores, as average of individual contexts scores, for the FNN in [11] (FNN) compared to the proposed LSTM, BLSTM and BLSTM with data augmentation (BLSTM+DA).

Method	$F1_{\text{AvgFram}}$	$F1_{\text{1-sec}}$
FNN [11]	58.4%	63.0%
LSTM	62.5%	63.8%
BLSTM	64.0%	64.6%
BLSTM+DA	**64.7%**	**65.5%**

The BLSTM network achieves better results than the FNN trained on the same data, improving the relative performance by 13.5% for the average framewise $F1$ and 4.3% for the 1 second block $F1$. The unidirectional LSTM network does not perform as well as the BLSTM network, but is still better than the FNN. The best results are obtained by the BLSTM network trained on the augmented dataset, which improves the performance over the FNN by relative 15.1% and 6.8% for the average framewise $F1$ and for the 1 second block $F1$ respectively.

Table 5.4: Results for each context in the dataset for the FNN in [11] (FNN), and our approach without data augmentation (BLSTM) and with data augmentation (BLSTM+DA).

	$F1_{\text{AvgFram}}$			$F1_{\text{1-sec}}$		
	FNN [11]	**BLSTM**	**BLSTM+DA**	FNN [11]	**BLSTM**	**BLSTM+DA**
basketball	70.2%	77.4%	**78.5%**	74.7%	79.0%	**79.9%**
beach	**49.7%**	46.6%	49.6%	**58.1%**	48.7%	51.5%
bus	43.8%	45.1%	**49.4%**	**52.7%**	47.3%	**52.7%**
car	53.2%	67.9%	**71.8%**	52.4%	66.4%	**69.5%**
hallway	47.8%	**58.1%**	54.8%	55.0%	**59.9%**	57.1%
office	77.4%	**79.9%**	74.4%	77.7%	**79.8%**	74.8%
restaurant	69.8%	76.5%	**77.8%**	73.7%	76.9%	**77.7%**
shop	51.5%	**61.2%**	61.1%	57.6%	60.9%	**61.7%**
street	62.6%	**65.3%**	65.2%	62.9%	63.3%	**63.9%**
stadium	58.2%	61.7%	**64.3%**	64.9%	64.2%	**66.2%**
average	58.4%	64.0%	**64.7%**	63.0%	64.6%	**65.5%**

In Table 5.4 we report the results for each context for the FNN in [11] (FNN), our BLSTM trained on the same data (BLSTM) and our BLSTM trained on the augmented data (BLSTM+DA). The results show that the proposed RNN, even without the regularisation from the data augmentation, outperforms the FNN in most of the contexts.

The $F1$-scores for different polyphony levels—reported in Table 5.5—are approximately the same, showing that the method is quite robust even when several events are combined.

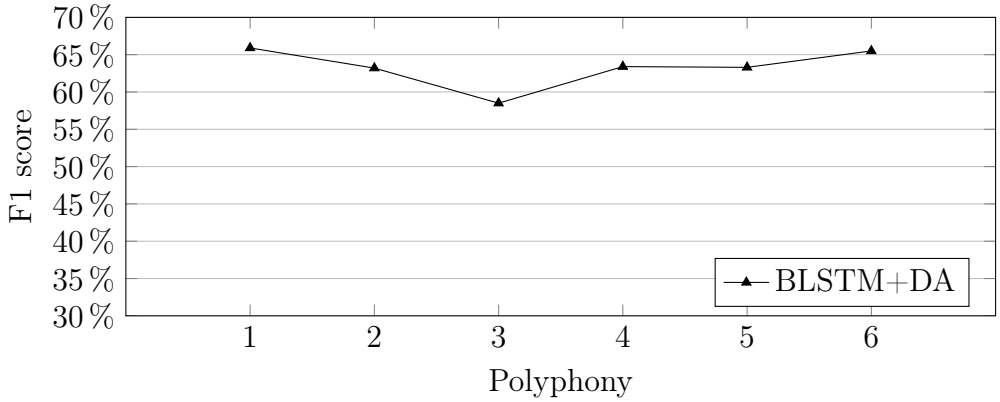

Figure 5.5: The $F1$ score computed at different polyphony levels for the BLSTM trained on the augmented dataset.

5.4.2 Artificial dataset

In Table 5.5 we report the scores of our BLSTM and LSTM networks trained on the original data, and BLSTM network trained with the proposed augmented techniques. As mentioned in Section 5.1.2, there is no published work conducted on this dataset. In order to compare our approach to the baseline, we evaluate a FNN similar to the one proposed in [11], using the same configuration and post-processing, changing only the number of neurons in the output layer to match the number of classes in this dataset.

Table 5.5: Overall $F1$ scores for the FNN using the approach in [11] (FNN), compared to the proposed LSTM, BLSTM and BLSTM trained also on the augmented data: BLSTM+TS (time stretching), BLSTM+SFTS (sub-frame time shifting), BLSTM+BM (block mixing), BLSTM+ALL (all the data augmentations).

Method	$F1_{\text{AvgFram}}$	$F1_{\text{1-sec}}$	$F1_{\text{Micro}}$
FNN	76.8%	74.7%	67.9%
LSTM	74.0%	71.5%	65.3%
BLSTM	78.4%	76.4%	69.9%
BLSTM+TS	77.9%	75.7%	69.5%
BLSTM+SFTS	78.4%	76.2%	70.0%
BLSTM+BM	**85.1%**	**84.4%**	**80.4%**
BLSTM+ALL	**85.2%**	**84.4%**	**80.5%**

The BLSTM network obtains slightly better results compared to the FNN trained on the same data. The unidirectional LSTM does not perform as well as either the BLSTM network nor the FNN. Concerning the BLSTM networks trained on the augmented datasets, both time stretching and sub-frame time shifting do not seem to improve the performance over the models trained on the standard dataset.

A large improvement derives from the block mixing technique. The BLSTM network trained on this augmentation improves the performance over the baseline approach using the FNN by relative 35.9%, 38.4% and 39.0% for the average frame-wise $F1$, 1 second block $F1$ and micro-average $F1$ respectively. Using all the data augmentation techniques together does not seem to improve the performance further.

When the performances of the different data augmentation techniques are compared, it should be considered that the block mixing technique in this dataset has the potential to create more novel observations that have higher polyphony. Moreover, it has been used to produce much more data than the time stretching and sub-frame time shifting techniques.

In Figure 5.6 we show the $F1$ scores at different polyphony levels for the FNN

and BLSTM trained on the original data, and the BLSTM trained also on the block mixing augmentation. The figure shows that the performance of both the FNN and the BLSTM trained only on the original dataset degrades very quickly as the polyphony level increases, with a slight advantage of the BLSTM. The BLSTM trained on the data from the block mixing is much more robust to higher polyphony observations, presenting a considerably less steep decrease in performance for higher polyphony levels.

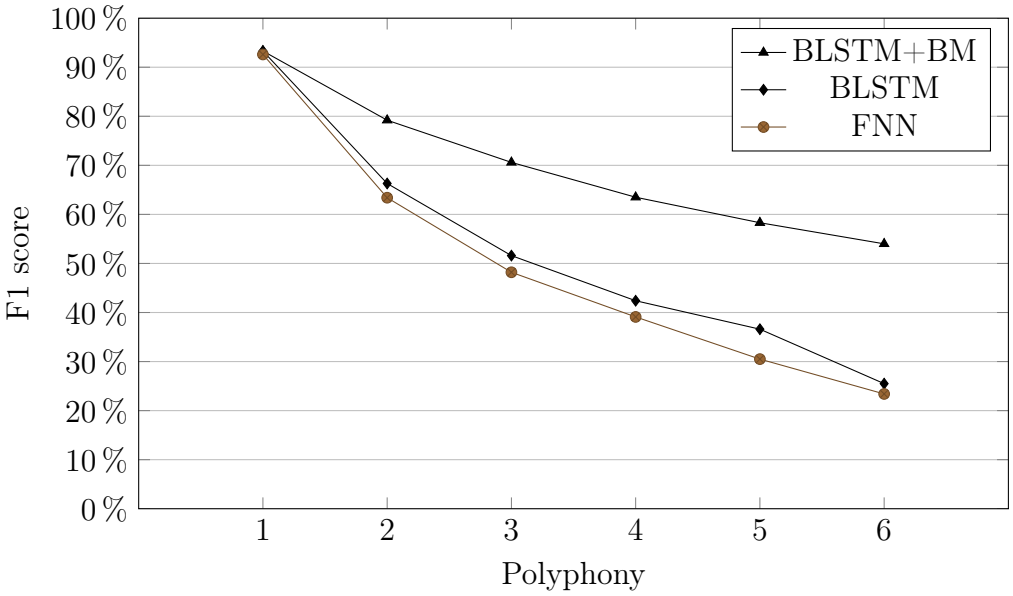

Figure 5.6: In AMD, the $F1$ score computed at different polyphony levels for the FNN and BLSTM trained on the original data, and for the BLSTM trained on the augmented dataset using block mixing (BLSTM+BM).

A representation of the outputs from the BLSTM+BM, the targets, the errors and the log mel energies for a 1 minute segment from AMD data is presented in Figure 5.7. As it can be seen from the Figure, the network correctly identifies most of the events with very good precision. The errors are mostly due to one or more undetected events in high polyphony segments.

5.5 Discussion

The RNNs outperform the baseline approach even when trained on the same data, despite the large amount of overfitting. It is interesting to notice that the RNNs have around 850K parameters each, about half of the 1.65M parameters in the FNN. The RNNs make a more efficient and effective use of the parameters, due to the recurrent connections and the deeper structure with smaller layers.

The block mixing technique largely improves the performances on the AMD, but not as much on the RLRD. This mismatch can be attributed to three factors. First,

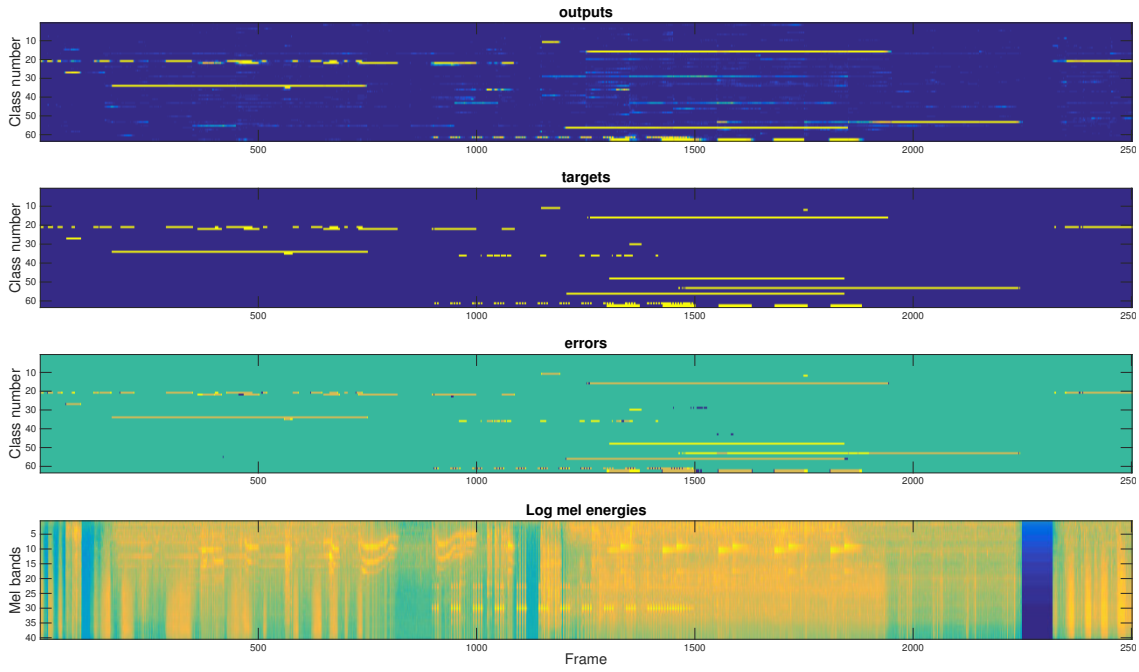

Figure 5.7: The first row represents the raw outputs from the network in a 1 minute segment, the second row the correct targets. Blue is zero, yellow is one. The third raw represents the TP (light brown), TN (turquoise green), FP (blue), FN (yellow), computed using the thresholded outputs and the targets. The last row contains the log mel energies of the segment.

in RLRD there are several long continuous events annotated for large sections of the recordings—such as *crowd walla*—which dominate all the metrics used in this work. Therefore, even if a system significantly improved the accuracy for short and isolated events, this would not emerge from these metrics.

Second, when two blocks are mixed inside the same context in RLRD, the resulting block might have—for a large part—the same classes present, due to the long sound events almost constantly active in the background. This makes the augmented data not as novel as in the case of AMD.

Last, since RLRD was manually annotated directly on the mixture of sounds collected from real environments, there are label inaccuracies that effectively limit the accuracy that can be reached by any system; this is not an issue in AMD, where the isolated recordings are automatically labeled and then randomly superimposed to simulate a continuous polyphonic recording from real life.

Concerning the performance of the other two data augmentation techniques, *i.e.*, time stretching and sub-frame time shifting, it is more difficult to draw conclusions from the results. While the two methods alone do not seem to improve significantly the performance, the amount of new data produced is small, especially considering

that it is highly correlated with the original data. A larger amount of data might be required to provide effective regularization.

Figure 5.2 shows that the major improvement from block mixing concerns higher polyphony segments of the data. This suggests that in order to recognize complex combinations of events, the network needs to be trained on mixtures of events rather than single events. This conclusion might apply to automatic music transcription as well, which is a domain similar to polyphonic SED.

Overfitting was the major issue encountered in training the networks. While the data augmentation techniques considerably helped generalization, all the networks showed significant overfitting. This suggests that more regularization—*e.g.*, in the form of dropout or more data augmentation—would possibly improve performance even more.

6. CONCLUSIONS

In this work we have proposed to use multilabel BLSTM recurrent neural networks for polyphonic sound event detection. RNNs can directly encode context information in the hidden layers and can learn the long time patterns naturally present in acoustic data. One of the main advantages of using RNNs for SED in real life environments is that they can be trained directly on a time-frequency representation of the polyphonic recordings. We have tested our proposed approach on two large datasets, outperforming on both the state-of-the-art FNN [11].

Moreover, since deep neural networks require large amount of data to avoid overfitting, we have tested three different data augmentation techniques in the context of polyphonic SED: time stretching, sub-frame time shifting and block mixing. The block mixing method—which allows to produce as much new data with higher polyphony as desired—largely improved the results on the second dataset. The improvement was particularly large on high polyphony segments of the test data.

Overall, for the first dataset—which contains real life recordings from 10 different contexts—our approach using RNNs and augmented dataset reports an average $F1$-score of 65.5% on 1 second blocks and 64.7% on single frames, a relative improvement over previous state-of-the-art approach of 6.8% and 15.1% respectively. For the second dataset—an artificial mixture of studio recordings—our system reports an average $F1$-score of 84.4% on 1 second blocks and 85.1% on single frames, improving over the baseline approach by 38.4% and 35.9% respectively. Moreover, the RNNs used have only half of the parameters of the baseline.

Due to the long events almost constantly active in the background of real life recordings, the metrics that rely on the events duration tend to neglect short events. A new metric should be designed, such that it equally rewards a system for detecting a long or short event, rather than basing its score solely on the duration.

Future work will concentrate on testing different data augmentation techniques, such as pitch shifting [103; 106], and more effective approaches to reduce overfitting. Dropout [58] in its form adapted to RNNs [116] is another regularization technique that has shown good results in reducing overfitting—such as in handwriting recognition using LSTM [117]—and might be applied to polyphonic SED as well.

Since collecting training data from real life environments is expensive and inevitably produces noisy labels, new strategies should be investigated. One pos-

sibility would be to use a coarser label resolution than frame-wise, such as using unsegmented labeled sequences of several seconds and connectionist temporal classification (CTC) [118]. Another approach might use a small labeled datasets and the large amount of unlabeled data available nowadays on the web, *e.g.* audio extracted from Youtube videos, for semi-supervised learning.

Concerning the model, further studies will develop on using attention mechanisms [119]—which have obtained excellent results in machine translation, automatically learning the alignment between features and labels—and extending RNNs by coupling them with convolutional neural networks, such as [120].

REFERENCES

[1] Yoshua Bengio, Ian J. Goodfellow, Aaron Courville, "Deep Learning", Book in preparation for MIT Press, 2015, URL: http://www.iro.umontreal.ca/~bengioy/dlbook.

[2] Richard F Lyon, "Machine hearing: An emerging field," in *Signal Processing Magazine, IEEE*, vol. 27, no. 5, pp. 131–139, 2010.

[3] Lawrence Rabiner, Biing-Hwang Juang, "Fundamentals of speech recognition". 1993, Prentice hall.

[4] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-Rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," in *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[5] Alex Graves, Abdel-rahman Mohamed, Geoffrey Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 2013, pp. 6645–6649.

[6] Jonathan William Dennis, "Sound event recognition in unstructured environments using spectrogram image processing", PhD thesis, Nanyang Technological University, Singapore, 2014.

[7] Lian-Hong Cai, Lie Lu, Alan Hanjalic, Hong-Jiang Zhang, Lian-Hong Cai, "A flexible framework for key audio effects detection and auditory context inference," in *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 14, no. 3, pp. 1026–1039, 2006.

[8] Min Xu, Changsheng Xu, Lingyu Duan, Jesse S Jin, Suhuai Luo, "Audio keywords generation for sports video analysis," in *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 4, no. 2, pp. 11, 2008.

[9] Selina Chu, Shrikanth Narayanan, CC Jay Kuo, "Environmental sound recognition with time–frequency audio features," in *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 17, no. 6, pp. 1142–1158, 2009.

[10] Aki Härmä, Martin F McKinney, Janto Skowronek, "Automatic surveillance of the acoustic activity in our living environment," in *IEEE International Conference on Multimedia and Expo (ICME)* 2005.

[11] Emre Cakir, Toni Heittola, Heikki Huttunen, Tuomas Virtanen, "Polyphonic Sound Event Detection Using Multi Label Deep Neural Networks," in *IEEE International Joint Conference on Neural Networks (IJCNN)* 2015.

[12] Annamaria Mesaros, Toni Heittola, Antti Eronen, Tuomas Virtanen, "Acoustic event detection in real life recordings," in *18th European Signal Processing Conference* 2010, pp. 1267–1271.

[13] Toni Heittola, Annamaria Mesaros, Tuomas Virtanen, Moncef Gabbouj, "Supervised model training for overlapping sound events based on unsupervised source separation," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 2013, pp. 8677–8681.

[14] Satoshi Innami, Hiroyuki Kasai, "NMF-based environmental sound source separation using time-variant gain features," in *Computers & Mathematics with Applications*, vol. 64, no. 5, pp. 1333–1342, 2012.

[15] Annamaria Mesaros, Toni Heittola, Onur Dikmen, Tuomas Virtanen, "Sound event detection in real life recordings using coupled matrix factorization of spectral representations and class activity annotations," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 2015, pp. 606–618.

[16] Florian Eyben, Sebastian Böck, Björn Schuller, Alex Graves, "Universal Onset Detection with Bidirectional Long Short-Term Memory Neural Networks.," in *International Society for Music Information Retrieval Conference (ISMIR)* 2010, pp. 589–594.

[17] Sebastian Böck, Markus Schedl, "Polyphonic piano note transcription with recurrent neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 2012, pp. 121–124.

[18] Giambattista Parascandolo, Heikki Huttunen, Tuomas Virtanen, "Recurrent neural networks for polyphonic sound event detection in real life recordings," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 2016 (under review).

[19] Weiwei Cheng, Eyke Hüllermeier, Krzysztof J Dembczynski, "Bayes optimal multilabel classification via probabilistic classifier chains," in *Proceedings of the 27th international conference on machine learning (ICML-10)* 2010, pp. 279–286.

[20] Shantanu Godbole, Sunita Sarawagi, "Discriminative methods for multi-labeled classification," in *Advances in Knowledge Discovery and Data Mining*, 2004, pp. 22–30.

[21] Michael Cowling, Renate Sitte, "Comparison of techniques for environmental sound recognition," in *Pattern Recognition Letters*, vol. 24, no. 15, pp. 2895–2907, 2003.

[22] Toni Heittola, Annamaria Mesaros, Antti Eronen, Tuomas Virtanen, "Context-dependent sound event detection," in *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2013, no. 1, pp. 1–13, 2013.

[23] Werner Heisenberg, "Über den anschaulichen Inhalt der quantentheoretischen Kinematik und Mechanik," in *Zeitschrift für Physik*, vol. 43, no. 3-4, pp. 172–198, 1927.

[24] Hyoung-Gook Kim, Thomas Sikora, "How efficient is MPEG-7 for general sound recognition?," in *Audio Engineering Society Conference: 25th International Conference: Metadata for Audio* 2004.

[25] Michael Kleinschmidt, "Localized spectro-temporal features for automatic speech recognition.," in *INTERSPEECH* 2003.

[26] RD Patterson, Ian Nimmo-Smith, John Holdsworth, Peter Rice, "An efficient auditory filterbank based on the gammatone function," in *A meeting of the IOC Speech Group on Auditory Modelling at RSRE*, vol. 2, no. 7, 1987.

[27] Stanley Smith Stevens, John Volkmann, Edwin B Newman, "A scale for the measurement of the psychological magnitude pitch," in *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, 1937.

[28] Steven B Davis, Paul Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," in *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.

[29] Douglas Reynolds, Richard C Rose, "Robust text-independent speaker identification using Gaussian mixture speaker models," in *IEEE Transactions on Speech and Audio Processing*, vol. 3, no. 1, pp. 72–83, 1995.

[30] Beth Logan, "Mel Frequency Cepstral Coefficients for Music Modeling," in *International Society for Music Information Retrieval Conference (ISMIR)* 2000.

[31] Michael A Casey, Alex Westner, "Separation of mixed audio sources by independent subspace analysis," in *Proceedings of the International Computer Music Conference* 2000, pp. 154–161.

[32] Shlomo Dubnov, "Extracting sound objects by independent subspace analysis," in *Audio Engineering Society Conference: 22nd International Conference: Virtual, Synthetic, and Entertainment Audio* 2002.

[33] Paris Smaragdis, Judith C Brown, "Non-negative matrix factorization for polyphonic music transcription," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics* 2003, pp. 177–180.

[34] Tuomas Virtanen, "Monaural sound source separation by nonnegative matrix factorization with temporal continuity and sparseness criteria," in *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 15, no. 3, pp. 1066–1074, 2007.

[35] Andrey Temko, Climent Nadeu, "Acoustic event detection in meeting-room environments," in *Pattern Recognition Letters*, vol. 30, no. 14, pp. 1281–1288, 2009.

[36] Arnaud Dessein, Arshia Cont, Guillaume Lemaitre, "Real-time detection of overlapping sound events with non-negative matrix factorization," in *Matrix Information Geometry*, 2013, pp. 341–371.

[37] Onur Dikmen, Annamaria Mesaros, "Sound event detection using non-negative dictionaries learned from annotated overlapping events," in *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)* 2013.

[38] Jonathan Dennis, Huy Dat Tran, Eng Siong Chng, "Overlapping sound event recognition using local spectrogram features and the generalised hough transform," in *Pattern Recognition Letters*, vol. 34, no. 9, pp. 1085–1093, 2013.

[39] Maureen Caudill, "Neural networks primer, part I," in *AI expert*, vol. 2, no. 12, pp. 46–52, 1987.

[40] Simon Haykin, "Neural networks and learning machines". vol. 3, 2009, Pearson Education Upper Saddle River.

[41] Xavier Glorot, Antoine Bordes, Yoshua Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics* 2011, pp. 315–323.

[42] John S. Bridle, "Training Stochastic Model Recognition Algorithms as Networks can Lead to Maximum Mutual Information Estimation of Parameters," in *Advances in Neural Information Processing Systems 2 (NIPS)*, 1990, pp. 211–217.

[43] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, *Learning internal representations by error propagation*, tech. rep., DTIC Document, 1985.

[44] Paul J Werbos, "Generalization of backpropagation with application to a recurrent gas market model," in *Neural Networks*, vol. 1, no. 4, pp. 339–356, 1988.

[45] Raúl Rojas, "Neural networks: a systematic introduction". 2013, Springer Science & Business Media.

[46] John Duchi, Elad Hazan, Yoram Singer, "Adaptive subgradient methods for online learning and stochastic optimization," in *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[47] Matthew D Zeiler, "ADADELTA: An adaptive learning rate method," in *arXiv preprint arXiv:1212.5701*, 2012.

[48] Yurii Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence O(1/k2)," in *Doklady an SSSR*, vol. 269, no. 3, pp. 543–547, 1983.

[49] Tijmen Tieleman, Geoffrey Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," in *COURSERA: Neural Networks for Machine Learning*, vol. 4, 2012.

[50] Diederik Kingma, Jimmy Ba, "Adam: A method for stochastic optimization," in *arXiv preprint arXiv:1412.6980*, 2014.

[51] Boris Teodorovich Polyak, "Some methods of speeding up the convergence of iteration methods," in *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[52] Xavier Glorot, Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International conference on artificial intelligence and statistics* 2010, pp. 249–256.

[53] Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)* 2012, pp. 1097–1105.

[54] Johan Håstad, "Computational limitations of small-depth circuits," 1987.

[55] Yoshua Bengio, Yann LeCun, "Scaling learning algorithms towards AI," in *Large-scale kernel machines*, vol. 34, no. 5, 2007.

[56] Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, Yoshua Bengio, "On the number of linear regions of deep neural networks," in *Advances in Neural Information Processing Systems (NIPS)* 2014, pp. 2924–2932.

[57] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, Samy Bengio, "Why does unsupervised pre-training help deep learning?," in *The Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[58] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," in *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[59] Bernhard Scholkopf, Alexander J Smola, "Learning with kernels: support vector machines, regularization, optimization, and beyond". 2001, MIT press.

[60] Yoshua Bengio, Olivier Delalleau, Clarence Simard, "Decision trees do not generalize to new variations," in *Computational Intelligence*, vol. 26, no. 4, pp. 449–467, 2010.

[61] Razvan Pascanu, Yann N Dauphin, Surya Ganguli, Yoshua Bengio, "On the saddle point problem for non-convex optimization," in *arXiv preprint arXiv:1405.4604*, 2014.

[62] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, Yann LeCun, "The loss surface of multilayer networks," in *arXiv preprint arXiv:1412.0233*, 2014.

[63] Kunihiko Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," in *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[64] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[65] Kurt Hornik, Maxwell Stinchcombe, Halbert White, "Multilayer feedforward networks are universal approximators," in *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[66] Razvan Pascanu, Guido Montufar, Yoshua Bengio, "On the number of inference regions of deep feed forward networks with piece-wise linear activations," in *arXiv preprint arXiv*, vol. 1312, 2013.

[67] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, Juergen Schmidhuber, "Deep big simple neural nets excel on handwritten digit recognition," in *arXiv preprint arXiv:1003.0358*, 2010.

[68] Abdel-Rahman Mohamed, George E Dahl, Geoffrey Hinton, "Acoustic modeling using deep belief networks," in *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 14–22, 2012.

[69] Yann LeCun, Yoshua Bengio, "Convolutional networks for images, speech, and time series," in *The handbook of brain theory and neural networks*, vol. 3361, no. 10, 1995.

[70] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, "Going deeper with convolutions," in *arXiv preprint arXiv:1409.4842*, 2014.

[71] Jeffrey L Elman, "Finding structure in time," in *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[72] Michael I Jordan, "Attractor dynamics and parallellism in a connectionist sequential machine," 1986.

[73] Alex Waibel, "Modular construction of time-delay neural networks for speech recognition," in *Neural Computation*, vol. 1, no. 1, pp. 39–46, 1989.

[74] Sepp Hochreiter, Jürgen Schmidhuber, "Long short-term memory," in *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[75] Kyunghyun Cho, Bart Merriënboer, Dzmitry Bahdanau, Yoshua Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *arXiv preprint arXiv:1409.1259*, 2014.

[76] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Yoshua Bengio, "How to construct deep recurrent neural networks," in *arXiv preprint arXiv:1312.6026*, 2013.

[77] Hava T Siegelmann, Eduardo D Sontag, "On the computational power of neural nets," in *Journal of computer and system sciences*, vol. 50, no. 1, pp. 132–150, 1995.

[78] Barbara Hammer, "On the approximation capability of recurrent neural networks," in *Neurocomputing*, vol. 31, no. 1, pp. 107–123, 2000.

[79] Paul J Werbos, "Backpropagation through time: what it does and how to do it," in *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

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

[81] Yoshua Bengio, Patrice Simard, Paolo Frasconi, "Learning long-term dependencies with gradient descent is difficult," in *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[82] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio, "On the difficulty of training recurrent neural networks," in *arXiv preprint arXiv:1211.5063*, 2012.

[83] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio, "Understanding the exploding gradient problem," in *Computing Research Repository (CoRR) abs/1211.5063*, 2012.

[84] James Martens, Ilya Sutskever, "Learning recurrent neural networks with hessian-free optimization," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)* 2011, pp. 1033–1040.

[85] Quoc V Le, Navdeep Jaitly, Geoffrey E Hinton, "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units," in *arXiv preprint arXiv:1504.00941*, 2015.

[86] Nicolas Boulanger-Lewandowski, Yoshua Bengio, Pascal Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," in *arXiv preprint arXiv:1206.6392*, 2012.

[87] Luca Pasa, Alessandro Sperduti, "Pre-training of Recurrent Neural Networks via Linear Autoencoders," in *Advances in Neural Information Processing Systems (NIPS)* 2014, pp. 3572–3580.

[88] Alex Graves, "Supervised sequence labelling with recurrent neural networks". vol. 385, 2012, Springer.

[89] Felix Gers, "Long short-term memory in recurrent neural networks", PhD thesis, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2001.

[90] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R Steunebrink, Jürgen Schmidhuber, "LSTM: A Search Space Odyssey," in *arXiv preprint arXiv:1503.04069*, 2015.

[91] Mike Schuster, Kuldip K Paliwal, "Bidirectional recurrent neural networks," in *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[92] Alex Graves, Jürgen Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," in *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005.

[93] Ilya Sutskever, Oriol Vinyals, Quoc VV Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems (NIPS)* 2014, pp. 3104–3112.

[94] Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, Jürgen Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 5, pp. 855–868, 2009.

[95] Alex Graves, "Generating sequences with recurrent neural networks," in *arXiv preprint arXiv:1308.0850*, 2013.

[96] Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan, "Show and tell: A neural image caption generator," in *arXiv preprint arXiv:1411.4555*, 2014.

[97] Wojciech Zaremba, Ilya Sutskever, Oriol Vinyals, "Recurrent neural network regularization," in *arXiv preprint arXiv:1409.2329*, 2014.

[98] Douglas Eck, Jurgen Schmidhuber, "Finding temporal structure in music: Blues improvisation with LSTM recurrent networks," in *Proceedings of the 2002 12th IEEE Workshop on Neural Networks for Signal Processing* 2002, pp. 747–756.

[99] Kiyotoshi Matsuoka, "Noise injection into inputs in back-propagation learning," in *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, no. 3, pp. 436–440, 1992.

[100] Patrice Y Simard, Dave Steinkraus, John C Platt, "Best practices for convolutional neural networks applied to visual document analysis". 2003,

[101] Dan Ciresan, Ueli Meier, Jürgen Schmidhuber, "Multi-column deep neural networks for image classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* 2012, pp. 3642–3649.

[102] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, "Going deeper with convolutions," in *arXiv preprint arXiv:1409.4842*, 2014.

[103] Navdeep Jaitly, Geoffrey E Hinton, "Vocal tract length perturbation (VTLP) improves speech recognition," in *Proc. ICML Workshop on Deep Learning for Audio, Speech and Language* 2013.

[104] Natsuki Kanda, Ryu Takeda, Yasunari Obuchi, "Elastic spectral distortion for low resource speech recognition with deep neural networks," in *IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)* 2013, pp. 309–314.

[105] Xiaodong Cui, Vikas Goel, Brian Kingsbury, "Data augmentation for deep neural network acoustic modeling," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* 2014, pp. 5582–5586.

[106] Jan Schlüter, Thomas Grill, "Exploring data augmentation for improved singing voice detection with neural networks," in *International Society for Music Information Retrieval Conference (ISMIR)* 2015.

[107] Brian McFee, Eric J. Humphrey, Juan P. Bello, "A Software Framework for Musical Data Augmentation," in *International Society for Music Information Retrieval Conference (ISMIR)* 2015.

[108] Olivier Lartillot, P Toiviainen, T Eerola, "MIRtoolbox," in *University of Jyväskylä*, 2008.

[109] Arthur Nádas, David Nahamoo, Michael Picheny, "Speech recognition using noise-adaptive prototypes," in *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 37, no. 10, pp. 1495–1503, 1989.

[110] MH Radfar, AH Banihashemi, RM Dansereau, A Sayadiyan, "Nonlinear minimum mean square error estimator for mixture-maximisation approximation," in *Electronics Letters*, vol. 42, no. 12, pp. 724–725, 2006.

[111] James L Flanagan, RM Golden, "Phase vocoder," in *Bell System Technical Journal*, vol. 45, no. 9, pp. 1493–1509, 1966.

[112] Toni Heittola, Annamaria Mesaros, Antti Eronen, Tuomas Virtanen, "Audio context recognition using audio event histograms," in *Proc. of the 18th European Signal Processing Conference (EUSIPCO)* 2010, pp. 1272–1276.

[113] Rainer Stiefelhagen, Keni Bernardin, Rachel Bowers, R Travis Rose, Martial Michel, John Garofolo, "The CLEAR 2007 evaluation," in *Multimodal Technologies for Perception of Humans*, 2008, pp. 3–34.

[114] Xi Zhou, Xiaodan Zhuang, Ming Liu, Hao Tang, Mark Hasegawa-Johnson, Thomas Huang, "HMM-based acoustic event detection with AdaBoost feature selection," in *Multimodal technologies for perception of humans*, 2008, pp. 345–353.

[115] Felix Weninger, "Introducing CURRENNT: The Munich Open-Source CUDA RecurREnt Neural Network Toolkit," in *Journal of Machine Learning Research*, vol. 16, pp. 547–551, 2015.

[116] Wojciech Zaremba, Ilya Sutskever, Oriol Vinyals, "Recurrent neural network regularization," in *arXiv preprint arXiv:1409.2329*, 2014.

[117] Vu Pham, Théodore Bluche, Christopher Kermorvant, Jérôme Louradour, "Dropout improves recurrent neural networks for handwriting recognition," in *IEEE 14th International Conference on Frontiers in Handwriting Recognition (ICFHR)* 2014, pp. 285–290.

[118] Alex Graves, Santiago Fernández, Faustino Gomez, Jürgen Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning* 2006, pp. 369–376.

[119] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, "Neural machine translation by jointly learning to align and translate," in *arXiv preprint arXiv:1409.0473*, 2014.

[120] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, Yoshua Bengio, "Attention-based models for speech recognition," in *arXiv preprint arXiv:1506.07503*, 2015.