DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Reconhecimento automático de cenas acústicas com técnicas de aprendizagem de máquina

Relatório submetido à Universidade Federal de Santa Catarina
como requisito para a aprovação da disciplina:
DAS 5511: Projeto de Fim de Curso

Gustavo Sena Mafra

Florianópolis, Agosto de 2016

Reconhecimento automático de cenas acústicas com técnicas de aprendizagem de máquina

Gustavo Sena Mafra

Esta monografia foi julgada no contexto da disciplina

DAS 5511: Projeto de Fim de Curso

e aprovada na sua forma final pelo

Curso de Engenharia de Controle e Automação

Prof. Daniel Coutinho

Banca Examinadora:

Quang-Khanh-Ngoc Duong
Orientador na Empresa

Prof. Daniel Coutinho
Orientador no Curso

Prof. Hector Bessa Silveira
Responsável pela disciplina

Prof. Marcelo Stemmer, Avaliador

Ariel Bruscatto, Debatedor

Guilherme Keiji Saito, Debatedor

Resumo

O relatório apresentado aqui é o resultado de um trabalho realizado entre 23 de Março de 2015 e 5 de Setembro de 2015 para a validação do estágio de final de curso, parte do curso de engenharia oferecido pela escola francesa Supélec e pelo *Master Recherche* (M2R) ATSI *(Automatique et traitement du signal et des images)*, oferecido conjuntamente pela Supélec, pela Université Paris-Sud 11 e pela ENS Cachan. Posteriormente, foi reutilizado como Projeto de Fim de Curso para o curso de Engenharia de Controle e Automação na UFSC (Universidade Federal de Santa Catarina).

Technicolor é uma empresa francesa, baseada em Issy-les-Moulineaux (arredores de Paris), antigamente conhecida como Thomson Inc. e Thomson Multimedia. Ela foi renomeada em 2010 para o nome de sua subsidiária americana. Os serviços/fontes de receita principais de Technicolor são serviços de entretenimento (para as indústrias do cinema e da televisão), as sua divisão *Connected Home* (responsável por set-top boxes, modems, dispositivos residenciais conectados) e patenteamento de tecnologias, este trabalho sendo relevante para os últimos dois.

O tema do estágio foi reconhecimento automático de cenas acústicas. Isto é, usar informações acústicas (sinais de audio) para inferir sobre o contexto dessa informação. Essa é uma forma particular de classificação de áudio, onde mais genericamente uma gravação de áudio é classificada em alguns rótulos pré-definidos. Exemplos de contextos/ambientes são ônibus, escritório, rua, etc.

O estágio serviu como uma revisão e ponto de entrada para pesquisa em classificação de audio na Technicolor. Um workflow completo foi implementado e uma grande variedade de métodos foram testados, com o objetivo de avaliar o potencial de diferentes features acústicas, métodos e estratégias de classificação, e abordagens gerais em aprendizagem de máquina.

Palavras-chave: aprendizagem de máquina, processamento de sinais, redes neurais, classificação de áudio.

Abstract

The report presented here is the result of a work realized between the 23 March 2015 and 04 September 2015 for the validation of the end-of-studies internship that is part of the engineering course of Supélec and for the *Master Recherche* (M2R) ATSI *(Automatique et traitement du signal et des images)* offered jointly by Supélec, Université Paris-Sud 11 and ENS Cachan. Posteriorly, it was also used as the final project of the Control and Automation Engineering course offered by UFSC (*Universidade Federal de Santa Catarina*).

Technicolor is a French company based in Issy-les-Moulineaux and previously known as Thomson Inc. and Thomson Multimedia. It rebranded itself in 2010 after its American film subsidiary. The main services/revenues of the company are entertainement services (for the film and television industries), its Connected Home activity (set-top boxes, modems, connected devices) and technology licensing, the work presented here being relevant to the last two.

The subject of the internship was Acoustic scene recognition. It consists in using acoustic information (audio signals) to infer the context of this information. It is a particular form of audio classification in which more generally an audio recording is classified in some predefined labels. Examples of such environments are bus, office, street, etc.

The internship served as a review and an entry point for research in Audio Classification at Technicolor. A complete workflow was implemented and a large variety of methods was tested, hoping to evaluate the potential of different audio features, classification methods and strategies and general machine learning approaches.

Keywords: machine learning, signal processing, neural networks, audio classification.

Lista de ilustrações

Lista de tabelas

Glossário

ANN Artificial Neural Network. 37, 39, 42

BRBM Binary-Binary Restricted Boltzmann Machine (RBM). 31, 32

CDBN Convolutional Deep Belief Network. 32, 71

CNN Convolutional Neural Network. 7, 9, 12, 32, 37, 39–42, 66, 67

CRBM Convolutional RBM. 32, 33

CUDA Compute Unified Device Architecture. 53

DBN Deep Belief Network. 31, 33, 68

DFT Discrete Fourier Transform. 26, 27

EBM Energy-Based Model. 29–31

EI Expected Improvement. 50

FFT Fast Fourier Transform. 26

GMM Gaussian Mixture Model. 19, 23

GPGPU General-Purpose computing on GPUs. 53, 54

GPU Graphical Processing Unit. 53

HMM Hidden Markov Model. 23, 42, 60

IDCT Inverse Discrete Cosine Transform. 27

IFT Inverse Fourier Transform. 27

MCMC Markov Chain Monte Carlo. 30

MFCC Mel-frequency Cepstrum Coefficient. 7, 9, 19, 21, 27, 28, 31, 32, 59, 68, 71

MLP Multi-layer Perceptron. 9, 33, 37, 38, 43, 61, 64, 65, 68, 71

PCA Principal Component Analysis. 27

RBM Restricted Boltzmann Machine. 11, 31–33, 68

RNN Recurrent Neural Network. 23

SGD Stochastic Gradient Descent. 44, 45, 61, 62, 66

STFT Short-term Fourier Transform. 26, 27, 59

SVM Support Vector Machine. 9, 19, 20, 22, 23, 33, 56–62, 71

TPE Tree of Parzen Estimators. 50, 57, 61–63, 65

Sumário

1 Introduction

The report presented here is the result of a work realized between the 23 March 2015 and 04 September 2015 for the validation of the end-of-studies internship that is part of the engineering course of Supélec and for the *Master Recherche* (M2R) ATSI *(Automatique et traitement du signal et des images)* offered jointly by Supélec, Université Paris-Sud 11 and ENS Cachan. Posteriorly, it was also used as the final project of the Control and Automation Engineering course offered by UFSC (*Universidade Federal de Santa Catarina*).

The choice of performing this internship in particular was motivated by the coherence of its subject with my choice of third-year at Supélec, where I attended the *majeure* MATIS *(Mathématiques appliquées au traitement de l'information et du signal)*, and also for its scientific character, presenting itself as a suitable for a student in M2R. Personally it also satisfies me at many levels, covering at the same time the fields of Signal Processing and Machine Learning.

Technicolor is a French company based in Issy-les-Moulineaux with strong research activity in Rennes (where the internship took place) and Hannover, Germany. Previously known as Thomson Inc. and Thomson Multimedia, it rebranded itself in 2010 after its American film subsidiary. The main services/revenues of the company are entertainement services (for the film and television industries), its Connected Home activity (set-top boxes, modems, connected devices) and technology licensing, the work presented here being relevant to the last two.

Technicolor's main interests in this work are developing technologies that could be used in the future in its Connected Home services, explore possibilities in a task that is not the main work area of the researchers and gain more mastery in the development aspects, applications and limitations of deep learning for audio, which is an approach that has been giving increasingly positive results in the last years for a large variety of tasks.

Inspired by this recent paradigm shift in image classification and automatic speech recognition, we focus on deep learning, trying to reduce as much as possible the use of hand-crafted application-specific feature extraction. This internship extrapolated a large variety of existent methods to a more challenging task where the amount of available labeled data is very limited as well as experimented with novel neural network structures and algorithms.

2 Problem Presentation

Increasingly, machines deployed in diverse environments can hear, whether they be mobile phones, hearing aids or autonomous robots. The problem in question is for them to make sense of what they hear.

Acoustic scene recognition consists in using acoustic information (audio signals) to infer the context of this information. It is a particular form of audio classification in which more generally an audio recording is classified in some predefined labels. Examples of such environments are bus, office, street, etc.

For intelligent systems to make best use of the audio modality, it is important that they can recognise not just speech and music, which have been researched as specific tasks, but also general sounds in an everyday typical indoor or outdoor environment.

Possible applications of acoustic scene recognition are smart homes and smart phones. For example, a smart phone could be set to automatically switch to silence mode when in a meeting or in a class or to increase the volume of its speaker in a noisy environment such as a bus or a metro. Smart homes could implement this as a security system that does not need to resort to video cameras which are more expensive and intrusive than microphones. It could be also be used in conjunction to images to automatically segment large videos such as movies.

3 Audio classification

3.1 General workflow

The general workflow of state-of-the-art systems for acoustic scene classification is usually divided into:

1. Feature extraction - baseline systems usually use pre-defined hand-crafted representations built upon the audio signal. The most used features are spectral-based: MFCC vectors [1] are obtained for short frames, then some statistics such as mean and variance of the many vectors for a whole recording are used for representing a signal. Other commonly used features are chroma, pitch, spectrograms, zero-crossing rate and linear predictive coding coefficients [2].

2. Classification - this task uses the features extracted by the previous step as the input for a general-purpose classification system. Commonly used models/algorithms are SVMs [3] and Gaussian Mixture Models (GMMs) [4].

This division is illustrated in Figure 1.

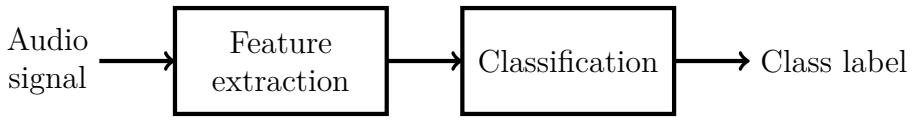

Figura 1 – Common general steps for performing audio classification.

Some advanced systems, to a greater extent outside the sub-field of acoustic scene recognition, proposed to learn the relevant features themselves. This is an intermediate step or a replacement for the first called feature learning [5]. The audio or the descriptors obtained in the feature extraction step are used to train a model that can learn more high level features from unlabeled data, as illustrated in Figure 2.

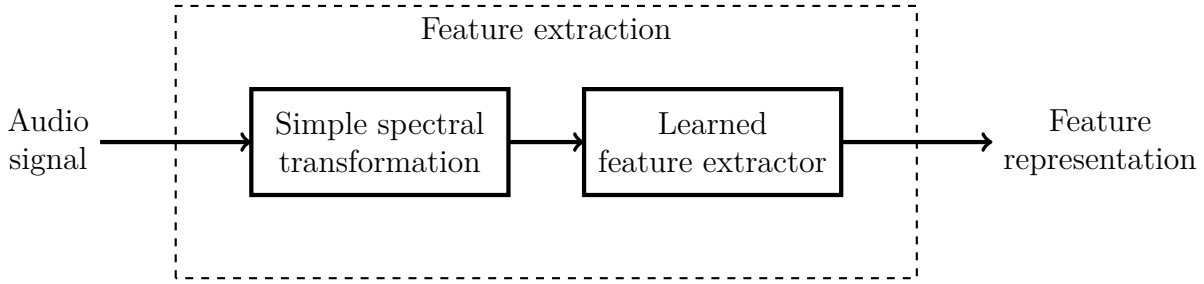

Figura 2 – Breaking down the feature extraction step in two sub-steps: optionally, one can separe it by one hand-crafted part (often a spectral-based transformation) and a learned feature extractor trained with audio data.

3.2 Limitations and challenges

The main problem in question is that acoustic scenes in general are not recognizable in the short-term. Therefore, it is reasonable to imagine that a large excerpt of audio, e.g. one to five seconds, would be needed in order to draw some reliable conclusions about it.

To make the use of classical classification algorithms such as an SVM feasible it is mandatory to transfer the signal to a representation where the instances could be more easily discriminated. To solve this problem most systems systematically discard hopefully irrelevant information while simultaneously calling upon creativity, intuition, or sheer luck to craft useful representations, gradually evolving complex, carefully tuned systems to address specific tasks [6], and despite considerable effort on feature extraction and years of research there is still no de-facto standard on it.

Discarding less information so to address a larger variety of sounds naturally increases the difficulty of the classification step in two ways: computationally (it takes longer to train the model and also to test and use it) and statistically (there must be more independent data to fit a more complex model).

One would then need:

1. A scalable framework to adapt to large amounts of data.
2. The data itself, that in the context of classification must be accompanied by metadata indicating the label of the class of scene that this recording belongs.

As imaginable it is not easy to find freely labeled data in the context of scene recognition for all the classes we want to detect (which is not the case for example in music classification where ID3 tags are widely available). Summarising, the solution must differ from speech and music classification by adopting more long-term representations and also by not relying solely on the availability of a great amount of labeled data. The ideal solution would be one that

1. is able to capture long-term patterns in the audio,
2. can generalize and extend to many environments,
3. disposes of a scalable framework for treating large amounts of information,
4. does not need a huge amount of labeled data for training, and
5. is not computationally overwhelming after training so that real-time applications become possible

3.3 Possible solutions

To capture long-term characteristics as well as instantaneous events that are typical of a scene it is common to use short-term frame-level spectral or cepstral descriptors and aggregate consecutive frame descriptors to a "super-frame" descriptor of a longer excerpt of audio [7, 8]. This process is shown in Figure 3.

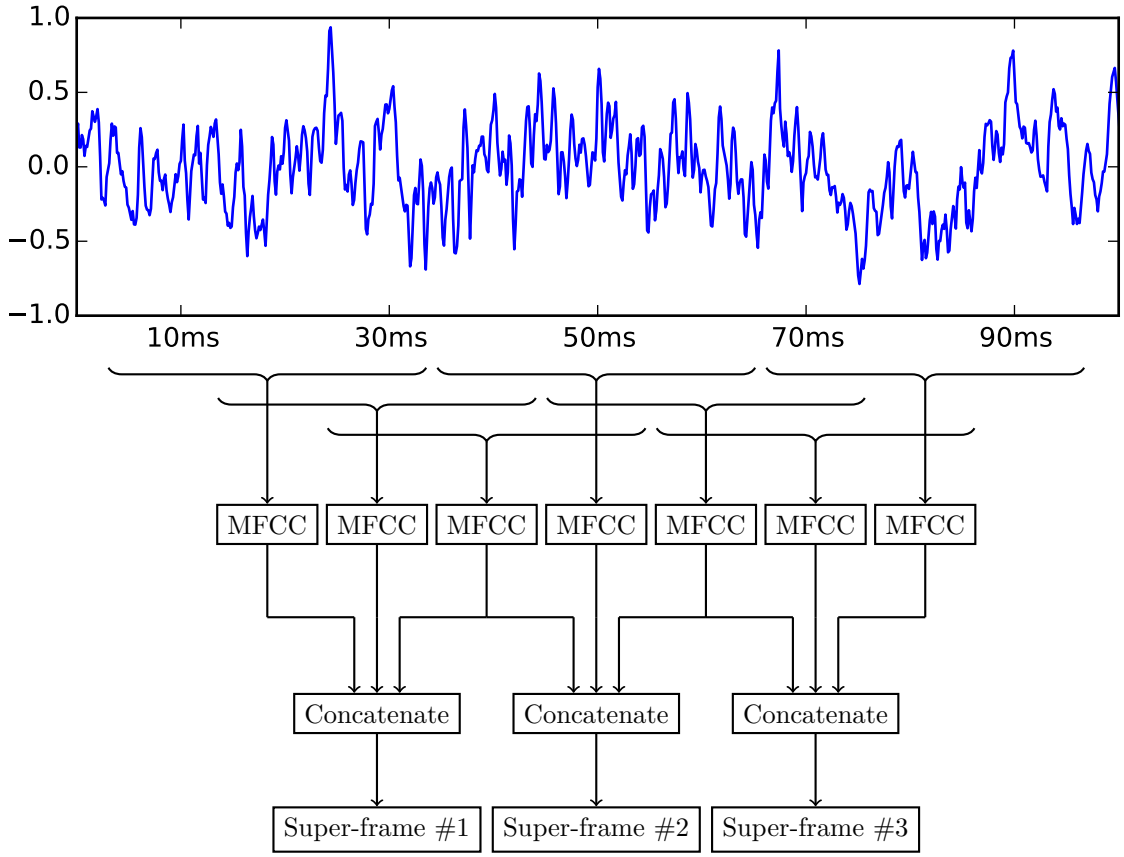

Figura 3 – Super-frame (or segment) construction for smaller frames. Despite the illustration, overlap between neighbour segments is typically greater than 50%. MFCCs are also an example, any frame descriptor could be used instead. Each of these segments is usually used as a classification instance.

To solve the issue of generalizability/extensibility, feature engineering can be abandoned here. The idea is to reduce this step as much as possible, taking the work out of human hands, and replacing it by a feature learning step.

Since labeled data are not widespread in the context of scene classification one idea would be to use two sets of data: the first one containing labeled or unlabeled data that in some way is similar to the audio we want to classify, which will be used for the representation learning step previously described. The second containing full labeled data, used to train the classifier. This approach has already been applied to speech and music classification [9] but apparently not to soundscapes.

Using the framework of neural networks/deep learning [10] we can explore its high flexibility and scalability, both for feature learning and classification. Testing/using a trained neural network is also known to be fast. The ability of interpreting the outputs of neural networks as probabilities also gives a mean for smoothly combining multiple excerpts into one without resorting to simple voting so that we can classify variable-length audio signals. SVMs are also an option that is particularly efficient against overfitting, but does not have probabilistic outputs.

3.4 Classifying signals of different lengths

When doing audio classification one normally works with features extracted on the frame level. In toy examples it is possible to separate the database between many frames, train the classifier and report accuracies on this level. However, in practical applications one could be more interested in assigning a label not on a frame or super-frame level, but on large excerpts that could be of variable length, such as songs.

A simple option is to extract frame-based features and for an entire excerpt compute the mean and/or the variance of these features. We would then be classifying on the "global" level. An example is shown in Figure 4.

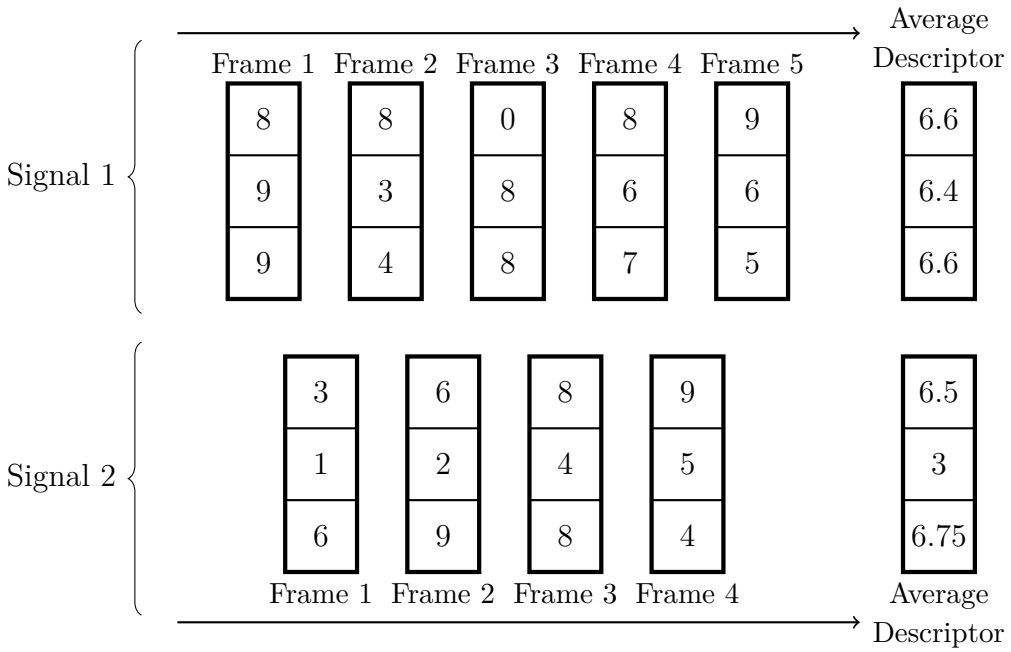

Figura 4 – Averaging signals of different lengths to a descriptor of unique size, allowing the use of classifiers. Each frame is described by a feature vector of length 3.

Another possibility to ultimately categorize an audio excerpt to one of the predefined classes is to ponderate the output probabilities given by the regressor for all segments in this excerpt. This strategy is known as the bag-of-frames approach [11]. More about this is discussed in Section 6.3.

It is also possible to mix these two, learning a feature extractor on the frame-level and a classifier on the global level for example.

3.5 Practical recognition: beyond classification

One important aspect for practical applications that is not being considered in the internship is the fact that a trained system, after deployment will often be devoted to perform real-time recognition. This is one step further from classification: the ultimate goal is not to classify static audio files, and the system will have to be able to detect changes in the environment.

The most common tool for this is using Hidden Markov Models (HMMs) on top of the classifier that now has the role of a regressor. The current state (that corresponds to a class label) of the system is then given by a Viterbi decoder that takes into account not only the last frame but a predicted state based on the previous frames. For an optimal performance of HMMs it is preferable to use systems that can effectively behave like a regressor, which is a disadvantage for SVMs.

Speech recognition has evolved from using GMM-HMM, later to deep neural networks combined with HMMs (DNN-HMM) and most recently to systems such as Recurrent Neural Networks (RNNs) that combine regression and prediction in the same structure. A review of these methods can be found in [12].

Although possible, it is not the best option to do real-time recognition with simple, static classification based on frames. A frame too big would introduce delays to the system and a frame too small could not be enough to correctly classify the audio.

4 Standard audio feature extraction

Feature extraction is motivated by the fact that machine learning tasks such as classification often require input that is mathematically and computationally convenient to process. However, real-world data such as audio and images are usually complex and redundant. Thus, it is necessary to discover useful features or representations from raw data.

In this section we precise the features/representations/descriptors used in this work on a conceptual level. All these features could be aggregated, averaged, or combined with other features.

4.1 Spectrogram

A spectrogram can be seen as a visual representation of the spectrum of frequencies in a sound or other signal as they vary with time or some other variable. In a common visualization of a spectrogram the frequencies go up in the vertical axis, and time is on the horizontal axis. A real example is shown in Figure 5.

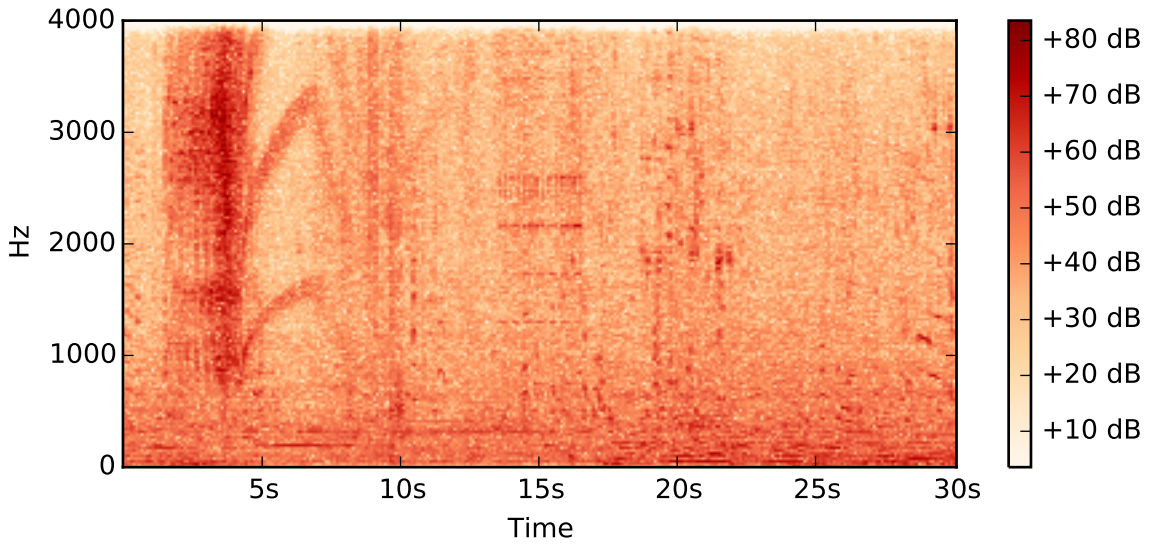

Figura 5 – Visual depiction of a spectrogram of a 30-second audio signal. The amplitudes are scaled logarithmically for a better visualization.

Mathematically, a discrete spectrogram is an $n_F \times n_W$ matrix, where n_W is the number of time windows used to obtain it and n_F is the number of bins used to represent the frequency domain.

This matrix can be obtained by computing Short-term Fourier Transforms (STFTs) of an audio signal and discarding the phase information, keeping only its absolute values.

The STFT of a discrete-time signal $x[n]$ is defined as

$$\mathrm{STFT}\{x[n]\}(m,\omega) \equiv X(m,\omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n} \tag{4.1}$$

where $w[n]$ is a window function (e.g. rectangular, triangular, Hann). In this case, m is discrete and ω is continuous, but the STFT is always performed on a computer using the Fast Fourier Transform (FFT), so both variables are discrete. The sum is also of course not infinite since audio recordings have a finite number of samples.

The spectrogram is then defined as:

$$\mathrm{Spectrogram}\{x(t)\}(m,\omega) \equiv |X(m,\omega)|^2 \tag{4.2}$$

In practice, the audio signal is divided into n_W short, overlapping frames. Typical values here are 25 ms frames and 10 ms overlaps. For each frame, we multiply the signal by a window function and we compute square of the absolute value of its Discrete Fourier Transform (DFT). Consecutive estimated spectrum vectors are concatenated to form a $n_F \times n_W$ matrix as previously described.

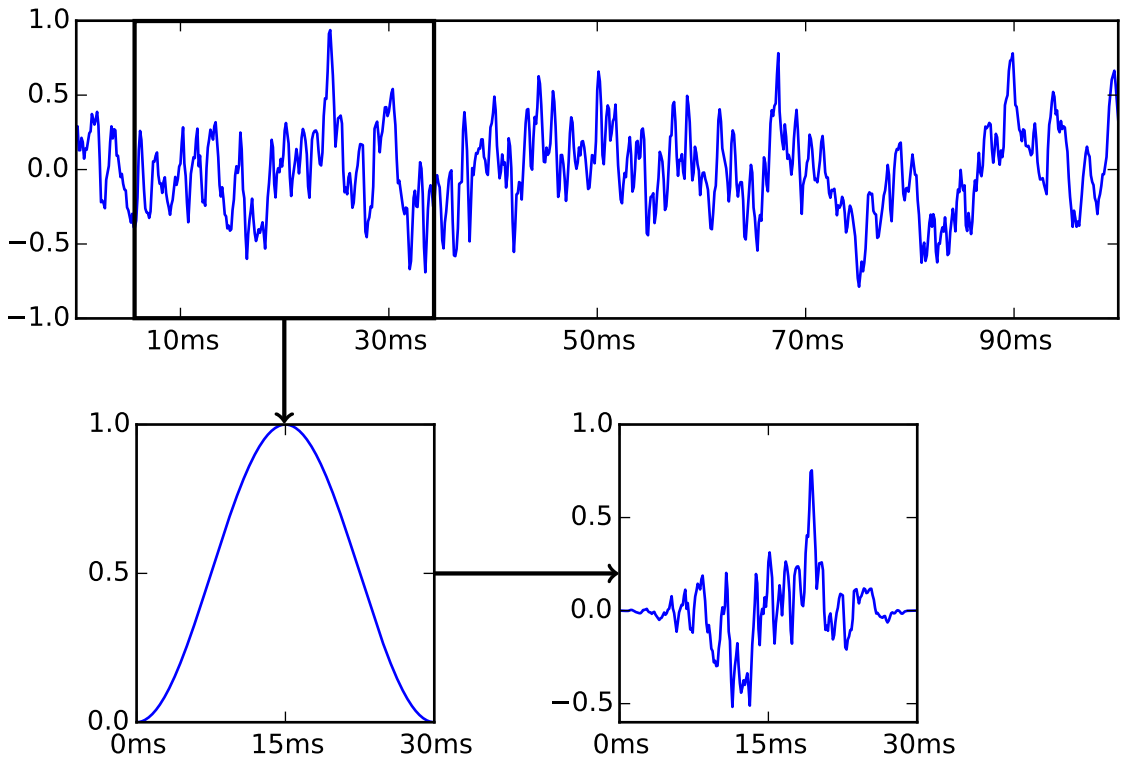

Figura 6 – Windowing of an audio signal and multiplication by a Hann window.

It is notably difficult to train certain classifiers with raw spectrograms because of its high dimensionality. For example, suppose we downsample our signal to 8 kHz and compute the STFT with 25 ms frames and 10 ms overlaps. Each DFT will contain $(0.025 \times 8000)/2 = 100$ frequency bins. The number of frames for a one-second audio excerpt will be equal to $1/0.01 = 100$. The spectrogram of this excerpt will then have 10000 scalars.

However, the dimensionality can be reduced by limiting its frequency or using algorithms such as Principal Component Analysis (PCA) [13], any non-linear high-level feature extractor, or both, as in [14]. Another option is to not aggregate consecutive vectors and use the frame descriptors themselves as classification instances, or to aggregate only a small number of frames.

4.2 Mel-frequency cepstrum

The cepstrum of a signal as it is defined in [15] is the result of taking the Inverse Fourier Transform (IFT) of the logarithm of the estimated spectrum of a signal:

$$\text{PowerCepstrum} = \left|\mathcal{F}^{-1}\left\{\log(|\mathcal{F}\{f(t)\}|^2)\right\}\right|^2 \tag{4.3}$$

The Mel-frequency cepstrum is the cepstrum with two modifications to the original cepstrum:

1. We do not use the IFT, but the Inverse Discrete Cosine Transform (IDCT), a similar transform as described in [16]. The IDCT has better properties for compression which is the goal of this step.

2. The inverse transform is evaluated on the spectrum transformed via the Mel scale, a perceptual scale of pitches judged by listeners to be equal in distance from one another.

A popular formula to convert f hertz into m mel is:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{4.4}$$

The scalars composing the cepstrum are then called Mel-frequency Cepstrum Coefficients (MFCC).

A great number of recent systems use MFCC-based features for classification. An MFCC vector is built from the localized DFT of a signal and therefore a group of MFCC vectors can be viewed as a higher-level of abstraction of a spectrogram.

Two important parameters for producing MFCCs are the number of coefficients and the number of filters when transforming the spectrogtam from the hertz scale to mel. With this one can adjust the dimensionality of the features obtained at the end of this step and consequently reduce the difficulty of the next processing step in the chain.

5 Feature learning with neural networks

Feature learning or representation learning [5] is a set of techniques that learn a transformation of raw or already transformed data input to a representation that can be effectively exploited in machine learning tasks.

Traditional hand-crafted features often require expensive human labor and often rely on expert knowledge. Also, they normally do not generalize well. This motivates the design of efficient feature learning techniques in the internship.

Feature learning can be divided into two categories: supervised and unsupervised feature learning. In this work we concentrate in unsupervised feature learning because of the high availability of unlabeled data while in practice it is very difficult to have large amounts of labeled data.

5.1 Energy-based models

Energy-Based Models (EBMs) [10] associate a scalar energy to each configuration of the variables of interest. Learning corresponds to modifying that energy function so that its shape has desirable properties. For example, we would like plausible or desirable configurations to have low energy. Energy-based probabilistic models define a probability distribution through an energy function, as follows:

$$P(\mathbf{x}) = \frac{e^{-Energy(\mathbf{x})}}{Z} \tag{5.1}$$

The normalizing factor Z is called the partition function, a constant to ensure that $P(\mathbf{x})$ sums or integrates to 1:

$$Z = \sum_{\mathbf{x}} e^{-Energy(\mathbf{x})} \tag{5.2}$$

with a sum running over the input space, or an appropriate integral when $\mathbf{x}$ is continuous.

In many cases of interest, we do not observe the example $\mathbf{x}$ fully, or we want to introduce some non-observed variables to increase the expressive power of the model. So we consider an observed part (still denoted $\mathbf{x}$) and a hidden part $\mathbf{h}$.

$$P(\mathbf{x}, \mathbf{h}) = \frac{e^{-Energy(\mathbf{x},\mathbf{h})}}{Z} \tag{5.3}$$

and because only $\mathbf{x}$ is observed, we care only about the marginal

$$P(\mathbf{x}) = \sum_{\mathbf{h}} \frac{e^{-Energy(\mathbf{x},\mathbf{h})}}{Z} \tag{5.4}$$

In such cases, to map this formulation to one similar to equation (5.1), we introduce the notation of free energy, defined as follows:

$$P(\mathbf{x}) = \frac{e^{-FreeEnergy(\mathbf{x})}}{\sum_{\mathbf{x}} e^{-FreeEnergy(\mathbf{x})}} \tag{5.5}$$

and $Z = \sum_{\mathbf{x}} e^{-FreeEnergy(\mathbf{x})}$, which means:

$$FreeEnergy(\mathbf{x}) = -\log \sum_{\mathbf{h}} e^{-Energy(\mathbf{x},\mathbf{h})} \tag{5.6}$$

The data log-likelihood gradient, quantity we will use to fit the unsupervised model, then has a particularly interesting form. We will again note θ as the set of all the parameters of the model. Differentiating equation (5.5) we obtain:

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = -\frac{\partial FreeEnergy(\mathbf{x})}{\partial \theta} + \frac{1}{Z} \sum_{\mathbf{x}} e^{-FreeEnergy(\mathbf{x})} \frac{\partial FreeEnergy(\mathbf{x})}{\partial \theta} \tag{5.7}$$

Again from equation (5.5) we can see that the probability of the data under the model appears in the last term:

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = -\frac{\partial FreeEnergy(\mathbf{x})}{\partial \theta} + \sum_{\mathbf{x}} P(\mathbf{x}) \frac{\partial FreeEnergy(\mathbf{x})}{\partial \theta} \tag{5.8}$$

The sum in the last term, or in continuous models an integral, is running over all possible combinations of the data. Although it is apparently difficult to compute its exact value, we can approximate it by observing that $\sum_{\mathbf{x}} P(\mathbf{x}) f(\mathbf{x})$ is nothing but the expected value of $f(\mathbf{x})$ under $P(\mathbf{x})$.

$$\frac{\partial \log P(\mathbf{x})}{\partial \theta} = -\frac{\partial FreeEnergy(\mathbf{x})}{\partial \theta} + \mathbb{E}_P \left(\frac{\partial FreeEnergy(\mathbf{x})}{\partial \theta} \right) \tag{5.9}$$

where $\mathbb{E}_P$ denotes the expected value under the models's distribution P. Therefore, if we could sample from P and compute the free energy easily, we would have a Monte-Carlo way to obtain a stochastic estimator of the log-likelihood gradient.

Analogously to the stochastic gradient descent algorithm described in section 6.2.3.1.2 we can also approximate the entire data log-likelihood by an estimate based on one sample.

$$\mathbb{E}_{\hat{P}} \left(\frac{\partial \log P(\mathbf{x})}{\partial \theta} \right) = -\mathbb{E}_{\hat{P}} \left(\frac{\partial FreeEnergy(\mathbf{x})}{\partial \theta} \right) + \mathbb{E}_P \left(\frac{\partial FreeEnergy(\mathbf{x})}{\partial \theta} \right) \tag{5.10}$$

Here, $\mathbb{E}_{\hat{P}}$ denotes expected value under the training set empirical distribution.

From this equation we can come up with most algorithms that are used to fit EBMs. The principle is to maximize the first term (the likelihood of the data) using a sample from the dataset (the second term) and a sample from the model (the third term) with a Markov Chain Monte Carlo (MCMC) technique.

5.2 Restricted Boltzmann Machines

RBMs, originally invented under the name Harmonium [17] are a particular type of EBM and the building blocks of Deep Belief Networks (DBNs) [18]. They are more constrained then their original counterpart, the Boltzmann Machine, but can be trained efficiently.

The restriction is that the units must form a bipartite graph: a pair of nodes from each of the two groups of units, commonly referred to as the "visible" and "hidden" units respectively, may have a symmetric connection between them, and there are no connections between nodes within a group. By contrast, "unrestricted" Boltzmann machines may have connections between hidden units.

RBMs are defined by their domain and their energy function (from which one derives its probability density), just like any other energy-based model, as described in equation (5.3). The energy function will take different forms depending on what we want to accomplish or model.

5.2.1 Binary-Binary RBM

This is the standard type of RBM, normally used to model gray-level images. Since MFCCs and other audio features do not adapt to a binary model this type of RBM is not used directly on these audio features, but only in higher levels of abstraction.

The energy function in a Binary-Binary RBM (BRBM) is

$$Energy(\mathbf{x}, \mathbf{h}) = -\mathbf{b}^T\mathbf{x} - \mathbf{c}^T\mathbf{h} - \mathbf{h}^T\mathbf{W}\mathbf{x} \tag{5.11}$$

This leads to the following free energy function:

$$FreeEnergy(\mathbf{x}) = -\mathbf{b}^T\mathbf{x} - \sum_i \log \sum_{h_i} e^{h_i W_i \mathbf{x}} \tag{5.12}$$

We can show that with this choice of energy function and considering the binary case (assuming the units take only the values 0 and 1) the probability of the hidden variables $\mathbf{h}$ given the visible variables $\mathbf{x}$ is

$$P(h_i = 1|\mathbf{x}) = \text{sigm}(W_i\mathbf{x} + c_i) \tag{5.13}$$

Since this energy function is symmetric, we can also write

$$P(x_j = 1|\mathbf{h}) = \text{sigm}(W_j^T\mathbf{h} + b_j) \tag{5.14}$$

For a Bernoulli distribution we know that its mean is equal to its probability of being equal one. Therefore, the conditional expectation of the hidden variables given the

visibles is

$$\mathbb{E}(h_i|\mathbf{x}) = \text{sigm}(W_i\mathbf{x} + c_i) \tag{5.15}$$

We can equally extend this to the whole vector of hidden units:

$$\mathbb{E}(\mathbf{h}|\mathbf{x}) = \text{sigm}(\mathbf{W}\mathbf{x} + \mathbf{c}) \tag{5.16}$$

which is exactly one of the typical activation functions in deterministic neural networks.

5.2.2 Gaussian-Binary RBM

To successfully fit a generative model to some data it is fundamental for it to be able to reconstruct this data. When working with MFCCs it is better to use a model that generates real gaussian visible units, in contrast to BRBMs. For more generality one can refer to [19], where RBMs used to model any input that follows a distribution belonging to the exponential family are described.

The energy function in a Gaussian-Binary RBM is [20]:

$$Energy(\mathbf{x}, \mathbf{h}) = -\frac{||\mathbf{b} - \mathbf{x}||^2}{2\sigma^2} - \mathbf{c}^T\mathbf{h} - \frac{\mathbf{h}^T\mathbf{W}\mathbf{x}}{\sigma^2} \tag{5.17}$$

With this choice of energy function and considering that the hidden variables are Bernoulli (they take only the values 0 and 1) we can show that $p(h_i = 1|\mathbf{x})$ continues to follow the equation (5.13), while the distribution of the visible units given the hidden units is gaussian:

$$P(x_j|\mathbf{h}) \sim \mathcal{N}(W_j^T\mathbf{h} + b_j, \sigma^2) \tag{5.18}$$

This type of RBM is notably more difficult to train then BRBMs. Once we learn the model we can use it to transform the data to a binary representation using equation (5.15) and stack several RBMs forming a deep network.

5.2.3 Convolutional RBM

Analogously to CNNs (Section 6.2.1.3) we can use Convolutional RBMs (CRBMs), a constrained version of an RBM that profits from locality, or we could see it as a probabilistic model of a CNN layer [21].

These can be stacked to form Convolutional Deep Belief Networks (CDBNs) [9]. Similarly to standard RBMs, the first layer should be chosen according to the distribution of the data.

5.3 Deep belief networks

DBNs are formed by stacking two or more RBMs on top of each other [18]. Similarly to supervised neural networks it is also possible to use CRBMs in the first layers so to profit from the local redundancy of data and use fully-connected RBMs at the top.

DBNs can be used in two similar ways:

1. As a pre-training strategy: deep supervised networks can be though to optimize starting from a random point. It often gets stuck in a plateau trying to adapt the upper layers while the lowest ones remain constant. This is known as the vanishing gradient problem [22]. Therefore, one can initialize a MLP (Section 6.2.1.2) with a DBN trained layer by layer whose expectation of the hidden units given the visible units matches with the activation function of the MLP. This initialization with DBN weights also acts as a regularization strategy, since it will have learned more robust transformations, independent from the classification task itself.

2. As a feature learner: this is more useful when the problem in question is the lack of labeled data. One can keep profiting from the advantages of deep architectures, using a high number of parameters, but breaks the network in two parts to avoid possible overfitting caused by the fact that the amount of labeled data is small. The lower part can be learned with unlabeled data as RBMs and the higher part is learned in a supervised way with backpropagation [23] or with SVMs to find a classification rule while the pre-trained part remains fixed.

5.3.1 Training algorithm

RBMs can be trained by using the Contrastive Divergence algorithm [24].

DBNs are trained layerwise: from the lowest RBM to the top, using the expectation of the hidden units as a fixed input for the upper RBM [25].

6 Classification methods

Classification is the problem of identifying to which class a new observation belongs based on a training set of data. In the context of this work, each observation will be a descriptor of an audio signal (a long excerpt or a short frame) and the classes are the environments or events we want to identify, e.g. bus, office, dog barking, etc.

In this section are described the main models used in this work and the algorithms used to train them.

6.1 Support vector machines

Support vector machines (SVM) are a class of classifiers that try to maximize a margin between different classes.

Originally developed as a linear classifier, we can apply SVMs to non-linear problems by projecting the original space in which the features lie in a larger space, where they could be linearly separable. SVMs contrast with neural networks this way because they don't transform the data to reach this state of linear separability.

Another notable characteristic of SVMs is that they only apply for the binary case. Thus, to extend the concept to multi-class classification one must perform certain tricks, often running multiple SVMs consecutively.

Despite this, SVMs are known for their ability to generalize across data, which is theoretically justified by having a low VC-dimension (for Vapnik–Chervonenkis dimension, a measure of the capacity of a classifier that establishes a probabilistic upper bound on the difference of the error in the training and test sets) [26].

6.1.1 Mathematical formulation

Here we will suppose all the data are labeled either as a positive or negative, and we note the label of the training data of index i as y_i, which will take values $+1$ and -1. To assign a value to each label corresponding to the data $\mathbf{x}_i$ we use the following rule:

$$\mathbf{x}_i\mathbf{w} + b \geq +1 \text{ for } y_i = +1 \tag{6.1}$$

$$\mathbf{x}_i\mathbf{w} + b \leq -1 \text{ for } y_i = -1 \tag{6.2}$$

This can be combined into one set of inequalities:

$$y_i(\mathbf{x}_i\mathbf{w} + b) - 1 \geq 0 \qquad \forall i \tag{6.3}$$

In the most simple case, when the samples are fully separable, the problem becomes simply to minimize $\|\mathbf{w}\|$ subject to the conditions previously stated, which corresponds to maximizing the margin between the hyperplanes separating the positive and negative samples.

To accomodate errors in the training data (i.e. when the data are not fully separable) we introduce slack variables ξ_i for each sample. The constraints become:

$$\mathbf{x}_i\mathbf{w} + b \geq +1 \text{ for } y_i = +1 - \xi_i \tag{6.4}$$

$$\mathbf{x}_i\mathbf{w} + b \leq -1 \text{ for } y_i = -1 + \xi_i \tag{6.5}$$

$$\xi_i \geq 0 \qquad \forall i \tag{6.6}$$

For one error to occur, the corresponding ξ_i must exceed unity. The sum of activated slack variables is then added as a penalty to the loss function:

$$\ell(\theta = \{\mathbf{w}, b, \xi\}, \mathcal{D}) = \frac{\|\mathbf{w}\|^2}{2} + C\left(\sum_i \xi_i\right)^k \tag{6.7}$$

where C is a parameter of regularization chosen by the user and k is normally one or two.

6.2 Neural networks

In this section we present neural network models used for classification, the loss function used to fit them and the algorithm used to search for optimal values for this loss function. Although each model is structurally very different, they have a lot in common, including the loss function and the optimization algorithm.

By neural network we mean a chain of non-linear transformations on data that can be composed of multiple layers. Each layer is composed of multiple units, that can also be called artificial neurons, by analogy with biological systems.

The output of a generic unit y_k, or artificial neuron is computed in the following way:

$$y_k = f\left(\sum_j w_{kj} x_j\right) \tag{6.8}$$

where x_j are input units (that can be data, scalar quantities or outputs from other units), w_{kj} are the weights (the parameters of a model) and f is a non-linear function called the activation function.

6.2.1 Network structures

In this section are described the different network structures used in this work: the logistic regressor, multilayer perceptrons and convolutional neural networks.

6.2.1.1 Logistic regressor

Although not a neural network, and chronologically preceding these models by decades [27], the logistic regressor is included in this section for many reasons:

1. it also consists of an affine transformation on the data followed by a non-linear transformation,
2. it is often used as the last layer of supervised neural networks (used for classification or regression),
3. it is normally optimized by maximizing the likelihood of the data given the parameters and optionally with aditional regularization terms, the same criteria used for MLPs and CNNs, and
4. the algorithm used to search for an optimal value for this loss function is normally a variant of the gradient descent method (the same for MLPs and CNNs).

The logistic regressor can be used as a probabilistic, linear classifier. It is parameterized by a weight matrix $\mathbf{W}$ and a bias vector $\mathbf{b}$. Classification is done by projecting an input vector onto a set of hyperplanes, each of which corresponds to a class. The distance from the input to a hyperplane reflects the probability that the input is a member of the corresponding class.

Mathematically, the probability that an input vector $\mathbf{x}$ is a member of a class i, a value of a stochastic variable Y, can be written as:

$$P(Y = i|\mathbf{x}, \mathbf{W}, \mathbf{b}) = \text{softmax}_i(\mathbf{W}\mathbf{x} + \mathbf{b}) = \frac{e^{W_i x + b_i}}{\sum_j e^{W_j x + b_j}} \tag{6.9}$$

The model's prediction y_{pred} is the class whose probability is maximal:

$$y_{pred} = \text{a}rgmax_i P(Y = i|\mathbf{x}, \mathbf{W}, \mathbf{b}) \tag{6.10}$$

To a logistic regressor or any other model here presented the reader is invited to refer to Section 6.2.2 and 6.2.3, where respectively the loss function and the algorithm to optimize it are presented.

6.2.1.2 Multilayer perceptron

The MLP, also known as an Artificial Neural Network (ANN) or a fully-connected neural network (in contrast with convolutional neural networks) or "vanilla" neural network is illustrated at Figure 7. It is the most common general-purpose deep architecture. By deep we mean that it is composed of multiple layers of non-linear transformations. It comes to aid when a logistic regressor is not able to capture all the patterns.

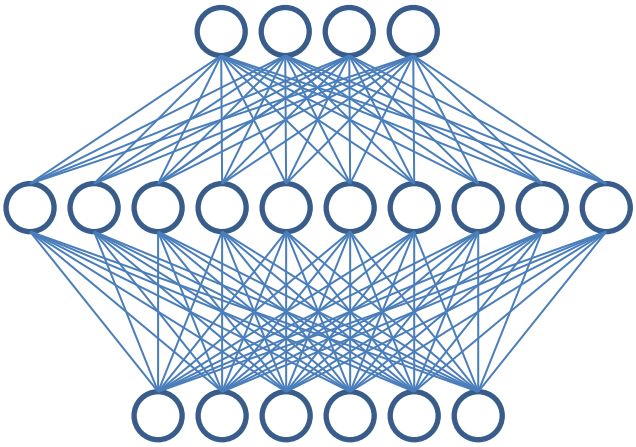

Figura 7 – Graphical representation of a fully-connected neural network with an input vector of size 6, 4 output classes and a single hidden layer of 10 units. An unit in an upper layer is activated depending on the values of the lower layer. Each connection between units has a weight value related to how these two units should behave at the same time. Increasing the number of hidden layers and units increases the capacity of representation of the network but requires more data to fit.

An MLP can be viewed as a logistic regression classifier where the input is first transformed using a learnt non-linear transformation. This transformation projects the input data into a space where it becomes linearly separable. This intermediate layer is referred to as a hidden layer. A single hidden layer is sufficient to make MLPs a universal approximator [28]. However there are substantial benefits to using many such hidden layers, i.e. the very premise of deep learning.

A typical set of equations for an MLP [23] is the following. Layer k computes an output vector $\mathbf{h}^k$ using the output $\mathbf{h}^{k-1}$ of the previous layer, starting with the input $\mathbf{x} = \mathbf{h}^0$,

$$\mathbf{h}^k = f(\mathbf{W}^k \mathbf{h}^{k-1} + \mathbf{b}^k) \tag{6.11}$$

with parameters $\mathbf{b}^k$ (a vector of offsets, or biases) and W^k (a matrix of weights). The function f is called the activation function and it is applied element-wise. Common options for activation functions are the sigmoid, the hyperbolic tangent and the rectified linear function.

The top layer output $\mathbf{h}^L$ is used for making a prediction and is combined with a supervised target y into a loss function. The output layer might have a non-linearity different from the one used in other layers, e.g. the softmax presented in equation (6.9). In this case, the last layer becomes a logistic regressor on a transformed version of the original data.

The complete set of equations for an ANN with two hidden layers would then be

$$\mathbf{h}^1 = f(\mathbf{W}^1\mathbf{x} + \mathbf{b}^1) \tag{6.12}$$

$$\mathbf{h}^2 = f(\mathbf{W}^2\mathbf{h}^1 + \mathbf{b}^2) \tag{6.13}$$

$$P(Y = i|\mathbf{x}, \Theta) = \text{softmax}_i(\mathbf{W}^L\mathbf{h}^2 + \mathbf{b}^L) \tag{6.14}$$

where Θ is the set of all parameters, i.e. $\{\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^L, \mathbf{b}^1, \mathbf{b}^2, \mathbf{b}^L\}$.

6.2.1.3 Convolutional neural networks

Originally developed for image processing, CNNs exploit from the redundancy and correlation between units [29]. They usually use a two-dimensional input, for example an image, and replace a full matrix multiplication as in the multilayer perceptron by a series of two-dimensional convolutions with trained kernels. This concept can be extended to one-dimensional inputs or tensors.

Although originally conceived for images it makes sense to use convolutional neural networks to classify any input given that they present locality properties, such as spectrograms or aggregated MFCC vectors. By locality properties we mean that neighbouring "pixels" in spectrograms are correlated.

6.2.1.3.1 Sparse connectivity

CNNs exploit spatially-local correlation by enforcing a local connectivity pattern between units of adjacent layers. In other words, the inputs of hidden units in an upper layer are from a subset of units in the lower layer, units that are spatially contiguous. The architecture thus ensures that the learnt kernels produce the strongest response to a spatially local input pattern.

However, stacking many such layers leads to (non-linear) kernels that become increasingly global (i.e. responsive to a larger region of pixel space in the case of images).

By reducing the connectivity between layers, CNNs have less parameters for the same number of hidden units and therefore one could argue that they are easier to train.

6.2.1.3.2 Shared weights

In addition, in CNNs, each kernel $\mathbf{h}_i$ is replicated across the entire lower layer. These replicated units share the same parameterization (weight vector and bias) and form a feature map.

Replicating units in this way allows for features to be detected regardless of their position in the visual field. Additionally, weight sharing also increases learning efficiency

by greatly reducing the number of free parameters being learnt. The constraints on the model enable CNNs to achieve better generalization on problems such as vision.

6.2.1.3.3 Mathematical formulation

In this section we continue to note the input units $\mathbf{x}$ and hidden units $\mathbf{h}$ as vectors. The extention to the two-dimensional case (images and spectrograms) is straight-forward.

We define a feature map $\mathbf{h}^k$ of dimension n_h as

$$\mathbf{h}^k = f(\mathbf{w}^k * \mathbf{x} + b^k) \tag{6.15}$$

where $\mathbf{w}^k$ are the weights and b^k is the bias. The convolution performed here is a valid convolution (i.e. no zero-padding). Therefore, if $\mathbf{x}$ is a vector of dimension n_x, $\mathbf{w}^k$ is a vector of dimension $n_w = n_x - n_h + 1$.

To form a richer representation of the data, each hidden layer is composed of multiple feature maps $\{\mathbf{h}^k, k = 1..K\}$, and therefore two consecutive layers are connected by multiple kernels $\{\mathbf{w}^k, k = 1..K\}$.

When stacking another convolutional layer that uses feature maps as inputs, the general formulation is to define a set of kernels for each combination of feature maps in the lower and upper layer. The complete set of weights for such a connection would then be a three-dimensional tensor. More precisely,

$$\mathbf{h}_2^k = f\left(\sum_j \mathbf{w}^{jk} * \mathbf{h}_1^j + b^k\right) \tag{6.16}$$

where $\mathbf{h}_2^k$ is the kth feature map in layer 2, $\mathbf{h}_1^j$ is the jth feature map in layer 1, and $\mathbf{w}^{jk}$ is the kernel associated with this connection.

In a two-dimensional setting each kernel becomes a matrix and the operation can be described by a four-dimensional tensor.

In practice it is also useful to treat input images as if they were a set of feature maps when using RGB. In this context, feature maps are called channels. This concept can also be applied to audio, treating frequency or frames as channels while the convolution occurs only over one of the axes.

6.2.1.3.4 Max-pooling

Another important concept of CNNs is max-pooling, which is a form of non-linear down-sampling. Max-pooling partitions the input image into a set of non-overlapping regions and, for each such of these regions, outputs the maximum value.

Max-pooling is useful for two reasons:

1. By eliminating non-maximal values, it reduces computation for upper layers.

2. It provides a form of translation invariance. This means that the same (pooled) feature will be active even when the image unergoes small translations.

Since it provides additional robustness to position, max-pooling is a "smart" way of reducing the dimensionality of intermediate representations.

6.2.1.3.5 CNNs for audio

CNNs applied on audio data, usually spectrograms, have a few notable characteristics. Since spectrograms can be viewed as images, one can apply CNNs the same way they are applied to gray-level images, but this is not usually done.

The usual way of applying CNNs to spectrograms is defining kernels over all the frequency range and therefore running a convolution over the time axis [14]. This is justifiable by the fact that a "pattern" detected in a high-frequency region is fundamentally different from this same pattern in a low-frequency region.

Since there is no convolution in the frequency axis, this can be viewed not as an axis, but as set of different channels, as previously described.

A full CNN applied to a spectrogram is illustrated in Figure 8.

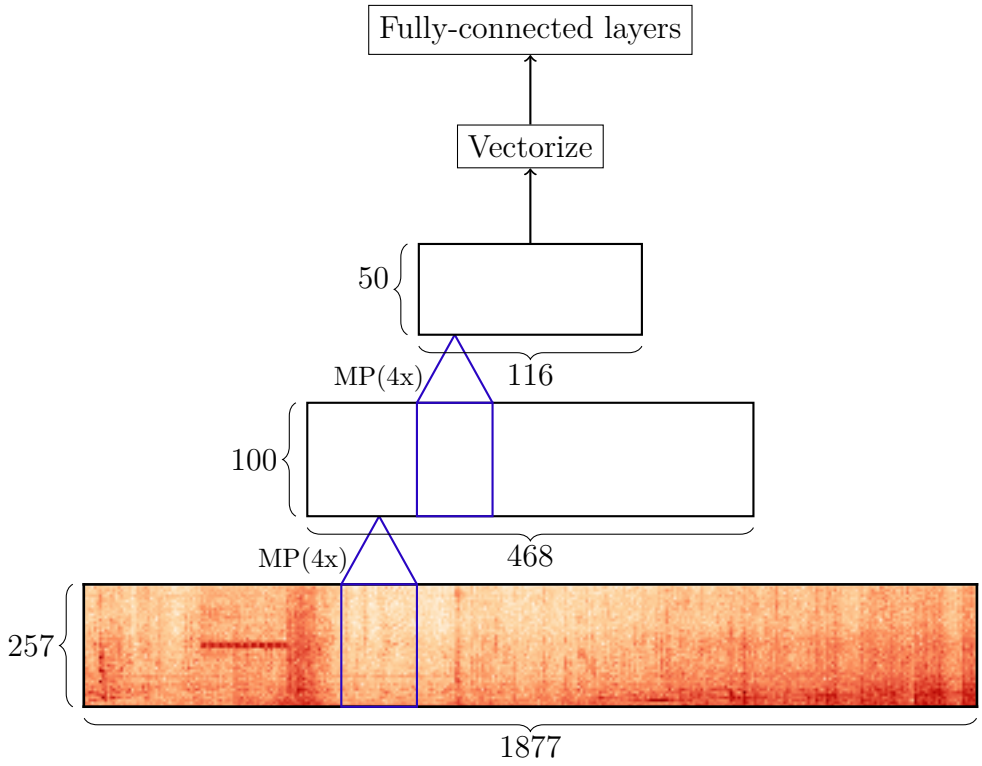

Figura 8 – CNN with two convolutional layers and any number of fully connected layers stacked on top of it. The input spectrogram has a high dimensionality that is reduced by the max-pooling steps of each layer. The dimension of the second layer for example depends on the max-pooling ratio (4) and the number of convolutional kernels (100) of the first layer.

An alternative way to apply CNNs on spectrograms is to define kernels ranging over all the time axis, performing a convolution on the frequency domain, especially in the context of automatic speech recognition when there is already the HMM that handles the temporal variations [30].

6.2.2 Loss function

Learning optimal model parameters involves minimizing a loss function. In the case of multi-class logistic regression or any ANN that uses a logistic regressor as its last layer, it is very common to use the negative log-likelihood as the loss. This is equivalent to maximizing the likelihood of the data set $\mathcal{D}$ under the model parameterized by θ.

The log-likelihood serves as a surrogate loss for the true cost we want to minimize (that being the error rate for all the instances). This error rate which we will call the 0-1 loss is not differentiable and has null gradient for practically all points. Thus, it would be impossible to work with gradient-based methods (Section 6.2.3).

Let us first start by defining the likelihood $\mathcal{L}$ and loss ℓ.

$$\mathcal{L}(\Theta|\mathcal{D}) = \prod_{i=0}^{|\mathcal{D}|} P(Y = y^{(i)}|\mathbf{x}^{(i)}, \Theta) \tag{6.17}$$

The log-likelihood is then:

$$\log \mathcal{L}(\Theta|\mathcal{D}) = \sum_{i=0}^{|\mathcal{D}|} \log(P(Y = y^{(i)}|\mathbf{x}^{(i)}, \Theta)) \tag{6.18}$$

where Θ represents all the parameters of the network. For example, for a single hidden layer MLP, $\Theta = \{\mathbf{W}_1, \mathbf{b}_1, \mathbf{W}_L, \mathbf{b}_L\}$. The loss function is defined as

$$\ell(\Theta, \mathcal{D}) = -\log \mathcal{L}(\Theta|\mathcal{D}) \tag{6.19}$$

The algorithm used to optimize this and other loss functions will be described in Section 6.2.3.

6.2.2.1 Regularization

It is often also necessary to add regularization penalties to this cost function to avoid overfitting. The most common ones are ℓ_1 and ℓ_2 penalties on the weight matrices or vectors. The loss function is changed to

$$\ell(\Theta, \mathcal{D}) = -\log \mathcal{L}(\Theta|\mathcal{D}) + \sum_{i=1}^{|\Theta|} \alpha_i|\theta_i| + \beta_i\theta_i^2 \tag{6.20}$$

where θ_i is an element from the set of parameters $\Theta = \{\theta_1, \theta_2, \theta_3, ...\}$. The regularization parameters can be different for each element from this set, or the same for example for all weight matrices. The bias vectors can be regularized but this is not usually done.

More modern strategies such as Dropout [31] can also be used. This however does not change the loss function, being a purely algorithmic regularizer.

6.2.3 Training algorithm

Training logistic regressors and ANNs is commonly done by gradient-based methods. Each network structure will have different parameters and thus different computations are required to compute the gradient for each of these structures. However, in a high level they all can be seen as implementing the same strategy.

Theano (Section 8.2), the library used for most of the implementation in this work has a feature of automatic differentiation. Therefore, the gradient does not need to be computed by hand. On top of that, Theano also automatically optimizes many computations which makes it very convenient to simply define and work with gradient-based algorithms in their most simple formulation, changing only the parameters and the forward path of the network when changing from one structure to another.

6.2.3.1 Gradient descent methods

Gradient descent is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the loss function at the current point.

6.2.3.1.1 Batch gradient descent

In batch gradient descent (the classical, standard gradient descent method), one computes the true loss function, based on all the available data for training:

$$\theta_{n+1} = \theta_n - \lambda_n \nabla_\theta \ell(\theta, \mathcal{D}) \tag{6.21}$$

where λ_n is the learning rate, or step-size and can be constant or updated according to a rule (Section 6.2.3.2).

6.2.3.1.2 Stochastic gradient descent

The principle of Stochastic Gradient Descent (SGD) is to replace the derivative of the loss function by an estimate of it. The simplest estimate is to evaluate this derivative but taking into account only one sample of the data. The update equation becomes

$$\theta_{n+1} = \theta_n - \lambda_n \nabla_\theta \ell(\theta, \mathcal{D}_i) \tag{6.22}$$

which is the same thing but on only one instance of the data. This update is then repeated for each instance $\mathcal{D}_i$. Of course, when using this strategy one must perform much more parameter updates than if using the classical, deterministic gradient descent.

In theory, the samples $\mathcal{D}_i$ to compute the gradient should be obtained randomly. However, it is expensive to compute a pseudo-number at each epoch and load these values from distant locations in the memory. Instead, it is better for computational reasons to sweep through the data as it is organized in the memory, always loading neighbour samples.

In practice datasets are almost always organized in a logical way. To simulate this random behavior we shuffle all the data in the memory before applying SGD, as illustrated in Figures 9 and 10.

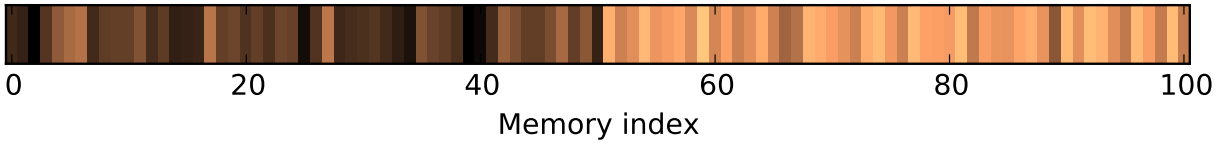

Figura 9 – Qualitative representation of a dataset stored in memory before being shuffled. Generally data is stored according to some internal organization and neighbour samples of data will be more correlated than distant ones as represented by the colors.

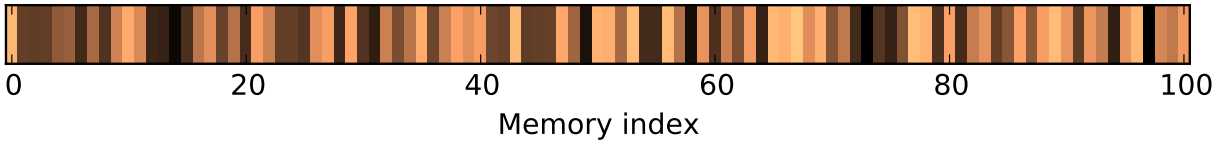

Figura 10 – The same data after being shuffled. With this we can simulate random sampling and profit from batch loading of the memory when sweeping through the data. This introduces the concept of epoch: each single sweep through the dataset is called an epoch.

An iteration through all the dataset, which is more comparable with an update of gradient descent shall be called an epoch and will be often used as a sort of checkpoint to investigate overfitting and apply early stopping rules and updating the learning rate.

Its advantages over gradient descent are:

1. It is faster to evaluate and therefore also to converge

2. It can explore more regions of the parameter space therefore avoiding local minima

6.2.3.1.3 Mini-batch stochastic gradient descent

A compromise between the two forms that computes the gradient against more than one training examples at each step. Each group of examples is called a “mini-batch”. This can perform significantly better than true SGD because the code can make use of vectorization libraries rather than computing each step separately. It may also result in smoother convergence, as the gradient computed at each step uses more training examples.

Recently with the use of graphical processing units for training such models, mini-batch gradient descent became even more popular because of the possibility of possible parallelizations.

6.2.3.2 Adaptive learning rate

To increase the speed of convergence one can adapt the learning rate of the gradient descent with AdaGrad [32]. It is a gradient-based method that attempts to "find needles in haystacks in the form of very predictive but rarely seen features". Given the udpate information from all previous iterations, the update formula proposed is as follows:

$$\theta_{n+1} = \theta_n - \alpha \frac{\nabla \ell(\theta_n)}{\sqrt{\sum_{n'=1}^{n} \left(\nabla \ell(\theta_{n'})\right)^2}} \tag{6.23}$$

Here we have replaced $\nabla_\theta \ell(\theta, \mathcal{D})$ by $\nabla \ell(\theta)$ for simplicity

6.2.3.3 Model selection

Gradient descent methods update the learner so as to make it better fit the training data with each iteration. Up to a point, this improves the learner's performance on data outside of the training set. Past that point, however, improving the learner's fit to the training data comes at the expense of increased generalization error.

To tackle this it is a common strategy in machine learning to separe the training data into two: the first continues to be used for gradient descent, and the second becomes a validation set, with which we will test the total loss at each epoch. The selected model is then the best one in the validation set between those found by the gradient descent applied on the training set. Ultimately, this model is then applied on a test set to give a prediction on its performance in a real-life application.

6.2.3.4 Random reinitialization and stopping criteria

The loss function being optimized is not convex. Therefore, we never know if it has converged to a global minima. To tackle this, it is common to run the optimiser multiple times, each time in a different initial point so it can converge to multiple local minima.

A simple criteria would be to make the (mini-batch) gradient descent to run for at least n_{me} epochs and $a * j_{best}$ epochs, where a is a predefined constant higher than one and j_{best} is the index of the best epoch. The best epoch could be measured either by the training loss (negative log-likelihood), the classification error or by a held-out validation data that does not participate in the computation of gradients or on the evaluation of results.

6.3 The bag-of-frames approach

To classify signals of various lengths we break them into constant-length segments as described in Section 3.4. To assign one unique label to the entire signal there are many

ways of weighting the information we have about its segments. Figure 11 illustrates the general process.

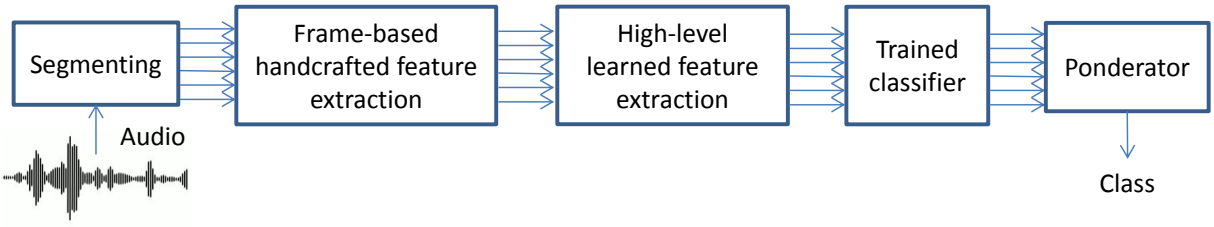

Figura 11 – Bag of frames

The first way is simple voting. Here we note $\mathcal{S}$ as the set of segments of the signal, y its class, and e_i the standard basis vector (all zero but one at the ith position).

$$p(y) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} e_{y_s} \tag{6.24}$$

A less brain-dead solution is to imagine each segment as an independent sample. The likelihood of the signal is then the product of its parts.

$$p(y) = p(y_1, y_2, ..., y_{|\mathcal{S}|}) = \alpha \prod_{s \in \mathcal{S}} p(y_s) \tag{6.25}$$

Although more justified by theory this option has the disadvantage of being overly sensible to outliers.

A third option would be a compromise between weighting probabilities and robustness, that is performing the sum of the distributions.

$$p(y) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} p(y_s) \tag{6.26}$$

Either way, the final class is computed as the maximum likelihood:

$$y_{pred} = \text{argmax}(p(y)) \tag{6.27}$$

7 Choice of hyper-parameters

There are several hyper-parameters in ANNs and CNNs, which are not (and, generally speaking, cannot be) optimized by gradient descent. Strictly speaking, finding an optimal set of values for these hyper-parameters is not a feasible problem. First, we can't simply optimize each of them independently. Second, we cannot readily apply gradient techniques that we described previously (partly because some parameters are discrete values and others are real-valued). Third, the optimization problem is not convex and finding a (local) minimum would involve a non-trivial amount of work.

However, over the last 30 years, researchers have devised various rules of thumb for choosing hyper-parameters in a neural network. A very good overview of these tricks can be found in [33].

A way to avoid using rules of thumb and "black art" techniques is to use algorithms that are specialized in optimizing black box functions. With recent advances in processing power it is now possible to run much more trials of a training and testing process to evaluate the performance of a set of parameters and thus it is at least feasible to improve classification results using such algorithms, often based on Bayesian optimization [34].

7.1 Bayesian optimization

In Bayesian optimization of hyper-parameters, the final generalization accuracy of a classifier, noted as α is modeled as

$$\alpha = f(\mathbf{X}, \mathbf{y}, \lambda, w) \tag{7.1}$$

where $\mathbf{X}$ is the training and validation data, $\mathbf{y}$ are the correct labels of each data sample, λ is the set of all the hyper-parameters and w is random noise.

The objective is to find an optimal λ_{opt} that maximizes the expected accuracy given a dataset:

$$\lambda_{opt} = \underset{\lambda}{\operatorname{argmax}} \left\{ \mathbb{E} \left(f(\mathbf{X}, \mathbf{y}, \lambda, w) | \mathbf{X}, \mathbf{y} \right) \right\} \tag{7.2}$$

The approach used to explore the parameter space of black box, noisy and expensive-to-evaluate functions such as the ones we are working with is different from classical optimization algorithms. Instead of choosing the next iteration in function of the current one, we choose it based on the entire observation history.

Typically this kind of algorithm can be put in the class of sequential model-based global optimization (SMBO) algorithms. A pseudo-code for an SMBO algorithm applied to a classifier black box function is presented in Algorithm 1.

Algorithm 1 SMBO(f, M_0, T, S)

1: $\mathcal{H} \leftarrow \varnothing$
2: **for** iteration $t < T$ **do**
3: Choose candidate λ_t based on the model M_t and a choice function S
4: Evaluate $\alpha_t = f(\lambda_t)$
5: $\mathcal{H} \leftarrow \mathcal{H} \cup (\alpha_t, \lambda_t)$
6: Update the model M_t based on the new observation history $\mathcal{H}$
7: **end for**

Eventhough rarely used in common optimization tasks, SMBO is a very general strategy with a few undefined characteristics. Here they are the choice function S, the model M (that will approximate f) and how it relates to the observation history $\mathcal{H}$.

The model M is usually a stochastic process but obviously not in the usual sense of a stochastic process that depends on time. Here it depends on a parameter space that can be continuous, discrete or mixed. More precisely, Gaussian processes (GPs) [35] have long been recognized as a good method for modeling loss functions because of some attractive properties.

A common choice function for a candidate to be evaluated is the Expected Improvement (EI) criterion:

$$\mathrm{EI}(\lambda) = \mathbb{E}_{M_t}\left\{\max(\alpha - \alpha^\star, 0)|\lambda\right\} \tag{7.3}$$

$$= \int_{-\infty}^{\infty} \max(\alpha - \alpha^\star, 0) p_{M_t}(\alpha|\lambda) d\alpha \tag{7.4}$$

where $\alpha^\star$ is a threshold that could be for example the best (higher) score obtained so far in the observation history $\mathcal{H}$. In the case of a classification task, α is an accuracy ratio, and the integral can be written as

$$\mathrm{EI}(\lambda) = \int_{0}^{1} \max(\alpha - \alpha^\star, 0) p_{M_t}(\alpha|\lambda) d\alpha \tag{7.5}$$

To conclude, at each iteration we choose the set of parameters that minimizes the EI criterion:

$$\lambda_t = \mathrm{argmax}_\lambda\, S(M_t, \lambda) \tag{7.6}$$

$$= \mathrm{argmax}_\lambda\, \mathrm{EI}(\lambda) \tag{7.7}$$

The hyper-parameter optimization algorithm we use in this work is named Tree of Parzen Estimators (TPE) [36]. It can be viewed as a EI optimizer but with some

particularities. Notably, it is able to model parameter spaces where at some locations, one set of hyper-parameters is known to be irrelevant, which can be quite useful when working for example with neural networks, where the number of hidden units in the second hidden layer is undefined when the network has only one hidden layer.

8 Development tools

An important aspect of machine learning and specially deep learning is the computation time required to train our models. Therefore one must take into account the computational aspect before chosing a programming language and scientific libraries.

8.1 GPU programming

To attain better performance with deep learning algorithms it is often necessary to do General-Purpose computing on GPUs (GPGPU), as shown in several available benchmarks. Modern Graphical Processing Units (GPUs) largely outperform modern CPUs on the great majority of tasks, profiting from parallelization and efficient matrix multiplication routines used in computer graphics.

Compute Unified Device Architecture (CUDA) is a parallel computing platform and API model created by NVIDIA. It allows software developers to use a CUDA-enabled GPU for general purpose processing. The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements. The CUDA platform is designed to work with programming languages such as C, C++ and Fortran.

C, C++ and Fortran are already low-level programming languages where one risks to pass more time to debug and to do safe code than to develop new ideas. C-CUDA complicates this even more, putting the programmer in a place where he will directly call GPU routines.

Therefore, there were considerable recent efforts to efficiently mask the C-CUDA or general GPU code in recent scientific computing libraries written in higher-level programming languages. The ones that were most successful and attained enough popularity in deep learning were Theano (written in Python) [37] and Torch (written in Lua) [38].

8.2 Python scientific stack

For our experiments Theano/Python was chosen, for having a richer environment and being more widespread in research (consequently having more community support). Theano is a numerical computation library. Computations are expressed using a NumPy-like syntax and compiled to run efficiently on either CPU or GPU architectures.

The predominant library for doing machine learning in Python is Scikit-learn, but not often used when doing neural networks and treating huge amounts of data, because of

issues with scalability. Scikit-learn functionalities use at the core NumPy and standard C, and it does not support GPGPU.

9 Experiments and results

The system was implemented and tested with the IEEE CASA database [39] which comprises 10 classes of acoustic scenes. The representation learning step was most often being performed on the UrbanSound database [40].

To compare our results on the IEEE CASA database with existent ones already published [2] we follow the same testing procedure: doing a 5-fold stratified cross-validation and evaluating results on audio files of 30 seconds.

More generally, k-fold cross validation works by dividing the development set in k folds. At each round, one takes one of the folds and holds it out of the training data to use it to estimate the accuracy of the classifier, and this classifier is trained by the remaining data. The final estimation of the accuracy of a classifier is made by the average of the obtained accuracy ratios for k different validation sets, namely, the k folds used to divide the data. Figure 12 illustrates this process.

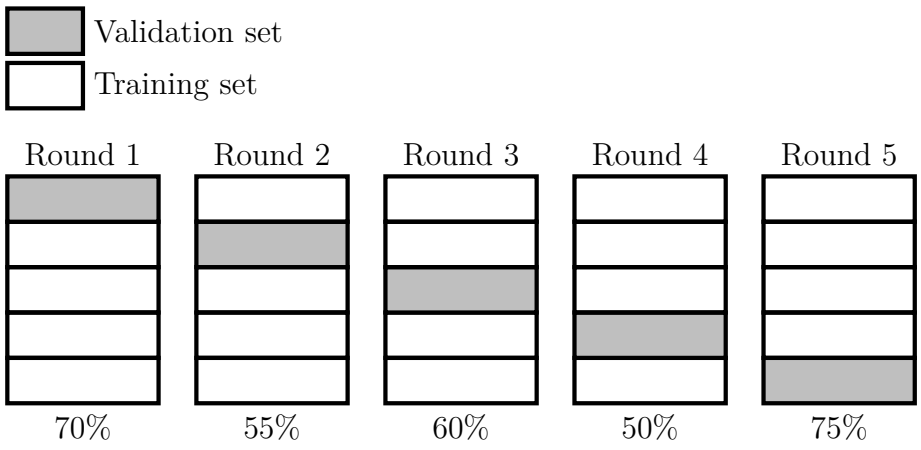

Figura 12 – 5-fold cross validation. The percentages displayed at the bottom of each bar are the accuracy ratios for the validation set of this round with a classifier trained by the training set of the same round.

9.1 Feature evaluation

This section presents a comparison between empirical results obtained from different features used in this work. As we said previously, we want to avoid complex structures and minimize the human labour in feature engineering, so we only evaluated simple frequency-based transformations which are often used.

9.1.1 Experimental setup

To compare these transformations we have fixed the overall strategy in a very simple form: we resample the audio signals to 8 kHz, we divide them by windowed frames, we compute the transformation for each of these frames and for each signal we compute the mean across all frames, resulting in a descriptor for the entire file.

Noting $\mathbf{y}_i$ the i^{th} frame of the signal, multiplied by a Hann window (Figure 6), and N the number of frames for each file, the following features were considered:

1. Spectrum: $\mathbf{x} = \frac{1}{N}\sum_i |\mathcal{F}(\mathbf{y}_i)|^2$
2. Log-spectrum: $\mathbf{x} = \frac{1}{N}\sum_i \log\left(|\mathcal{F}(\mathbf{y}_i)^2|\right)$
3. Mel-log-spectrum: $\mathbf{x} = \frac{1}{N}\sum_i \log\left(M\,|\mathcal{F}(\mathbf{y}_i)^2|\right)$, M being the transformation matrix from Hertz to the Mel scale.
4. Mel-frequency cepstrum coefficients: $\mathbf{x} = \frac{1}{N}\sum_i \left|\mathcal{F}^{-1}\left\{\log\left(M\,|\mathcal{F}(\mathbf{y}_i)^2|\right)\right\}\right|^2$

These descriptors are ordered in increasing order of complexity, but for these experiments, all of them contain the same amount of information (ignoring possibilities such as a non-inversible transformation squared matrix), i.e. they all have the same dimension, that depends on the window size.

We consider here the "best" feature as the one that results in the best accuracy ratios for a type of classifier. We chose for this step to use a linear SVM as a classifier due to its simplicity, reduced number of hyper-parameters (just one, the regularization parameter C) and its speed of computation.

Before feeding the data into the classifier, we also normalize all the data for mean 0 and unit variance based on the training set. Denoting T the training set and V the validation set this operation can be described as follows:

$$\mu = \sum_{i \in T}\sum_j x_{i,j} \tag{9.1}$$

$$\sigma^2 = \sum_{i \in T}\sum_j (x_{i,j} - \mu)^2 \tag{9.2}$$

Where $\mathbf{x}_i$ is the vector associated with the i^{th} file in the training set and $x_{i,j}$ is its j^{th} feature (which is equivalent to a frequency bin for example in the spectrum).

The same affine transformation is then applied to both sets:

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu}{\sigma} \quad \forall i \in T \cup V, \quad \forall j \tag{9.3}$$

9.1.2 Finding good candidates

For each descriptor we run an hyper-parameter optimizer (TPE, section 7) for the size of the window used to compute the frames (w_{len}), the gap between windows (h_{len}, for hop length) that corresponds to the overlap and also the regularization parameter C of the SVM.

The TPE algorithm takes as an input an interval and a type of distribution for each parameter, more precisely we specify a prior distribution over our parameter space. For these experiments we have used:

$$w_{len} \in \mathbb{N} \sim \mathcal{U}(100, 10000) \tag{9.4a}$$

$$h_{len} \in \mathbb{N} \sim \mathcal{U}(50, 5000) \tag{9.4b}$$

$$C \in \mathbb{R} \sim \ln\mathcal{N}(\mu = 0, \sigma = 1) \tag{9.4c}$$

The parameter space of w_{len} and h_{len} is described in terms of samples, the sampling frequency being $f_s = 8\,\text{kHz}$.

It is also important to note that at each run we change the random seed used to make the sets that will be used for making the cross-validation folds. Final results are sensible to the way we divide folds and this way we reduce the bias of the hyper-parameter optimizer towards a configuration that would be better for a specific fold division, even if increasing the variance of the results.

After computing 100 different combinations of hyper-parameters for each type of descriptor we present the 10 systems that achieved the best accuracy ratios for each of these descriptors, along with the mean and median of these selected systems.

We note here acc_v the average accuracy obtained in the validation sets for each round, acc_t the same for the train sets, t_{fe} the total computation time for extracting the features from the audio signal (except the resampling step) and t_{tv} the total computation time for training and testing the SVMs used in all 5 rounds.

The results are displayed in Tables 1 to 4.

	1	2	3	4	5	6	7	8	9	10
w_{len}	110	720	120	1150	240	1000	1140	1090	110	680
h_{len}	470	1180	1630	2320	790	2560	1950	1050	1650	2980
C	2.22	1.45	1.69	1.59	2.28	1.45	1.38	3.83	1.79	1.14
acc_v	46%	45%	44%	44%	43%	43%	43%	43%	42%	42%
acc_t	62.5%	73.25%	63.25%	74.75%	69.5%	75.25%	73.75%	78.5%	62%	73.75%
t_{fe}(s)	0.67	0.86	0.54	0.91	0.71	0.73	0.96	2.58	0.53	0.67
t_{tv}(s)	28.02	58.72	28.59	80.54	34.90	73.88	80.42	76.31	27.89	56.94

Tabela 1 – 10 best systems (out of 100) based on the spectrum, using an SVM and classifying by files (average of all frame descriptors)

	1	2	3	4	5	6	7	8	9	10
w_{len}	2410	3360	3860	5640	1740	3140	3180	3060	9440	2060
h_{len}	120	540	770	3210	1500	2980	1490	1120	3510	260
C	33.05	0.46	0.94	1.31	24.33	2.11	0.44	82.52	1.30	0.80
acc_v	76%	74%	73%	72%	71%	71%	71%	71%	71%	71%
acc_t	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
t_{fe}(s)	82.21	8.25	16.70	3.47	2.14	3.63	4.09	4.49	5.54	18.66
t_{tv}(s)	142.73	191.68	214.77	302.76	110.22	178.41	181.33	172.86	510.52	124.98

Tabela 2 – 10 best systems (out of 100) based on the log-spectrum, using an SVM and classifying by files (average of all frame descriptors)

It is interesting to note the great jump of accuracy ratios from systems using a pure spectrum on a linear scale (Table 1) and a logarithmically-scaled spectrum (Table 2).

Furthermore, the accuracy of the predictor can reach up to 100% on the training set without the use of kernels for the SVM applied on the log-spectrum. This could raise doubts on whether a more complex classifier would be necessary for this case.

	1	2	3	4	5	6	7	8	9	10
w_{len}	4580	2810	4260	7970	1620	7000	7380	3140	5600	7500
h_{len}	3220	1570	2220	4280	800	2780	3050	4090	3110	2980
C	0.98	0.13	0.34	0.36	0.67	0.39	0.90	0.95	0.21	0.54
acc_v	75%	74%	74%	74%	73%	73%	73%	72%	72%	72%
acc_t	100%	98%	100%	100%	100%	100%	100%	100%	100%	100%
t_{fe}(s)	32.73	25.62	32.23	97.45	11.92	70.65	75.77	16.78	46.05	76.32
t_{tv}(s)	245.93	155.53	225.69	412.32	97.30	370.99	387.77	171.29	295.52	391.04

Tabela 3 – 10 best systems (out of 100) based on the mel-log-spectrum, using an SVM and classifying by files (average of all frame descriptors)

	1	2	3	4	5	6	7	8	9	10
w_{len}	7950	6100	9320	8070	8310	5970	6270	6280	7980	9090
h_{len}	3900	4350	4500	3770	1090	4630	3780	3800	3180	4450
C	14.90	3.46	1.89	13.75	13.30	9.89	16.30	17.85	13.65	1.93
acc_v	76%	74%	74%	73%	73%	72%	72%	72%	72%	72%
acc_t	100%	98%	98.25%	100%	100%	100%	100%	100%	100%	97.25%
t_{fe}(s)	206.06	125.82	269.37	219.91	358.37	120.91	132.76	138.42	219.94	246.94
t_{tv}(s)	359.03	322.10	417.98	363.60	373.30	269.08	329.42	332.73	355.80	436.11

Tabela 4 – 10 best systems (out of 100) based on MFCCs, using an SVM and classifying by files (average of all frame descriptors)

The results obtained with the mel-log-spectrum (Table 3) and MFCC (Table 4) do not raise the validation accuracy ratios compared to a more simple descriptor (the log-spectrum).

These two descriptors also have a high computation time for extracting the features. The implementation of these transformations was not particularly optimized, but with increased complexity and with no gains in performance, the use of these two descriptors may be not necessary.

An interesting thing to see here would be that the parameter regions or w_{len}, h_{len} and C that attained the best results, especially with MFCCs are different: using MFCCs seem to favour the use of large windows for computing the STFT and a high C parameter for the SVM, which means that the SVM doesn't need a lot of regularization.

9.1.3 Reducing the variance

To achieve a final and more reliable conclusion about what would be the best feature and system (between those using an SVM and taking the average frame descriptor as a classification instance), we select for each type of descriptor the one with the best accuracy ratio, the mean and the median of the best 10, totalizing 12 systems.

We compute the cross-validation estimate of the accuracy ratio for 10 different fold divisions to reduce the variance of this estimator. The random seed used to separate the folds was in this case the same for each system to avoid variance caused by the different divisions and to bias all systems in the same way. The results are displayed in Table 5.

Descriptor	w_{len}	h_{len}	C	acc_v	acc_t	t_t (s)	t_f (s)
Spectre	110	470	2.22	42.65%	62.42%	0.34	0.61
	636	1658	1.88	41%	74.62%	0.86	0.98
	680	2980	1.14	42.55%	73.87%	0.92	0.65
Log-spectre	2410	120	33.05	69.95%	100%	2.51	90.378
	3789	1550	14.73	68%	100%	3.76	18.12
	3160	1305	1.31	68.05%	100%	3.27	5.22
Mel-log-spectre	4580	3220	0.98	69.15%	100%	4.46	36.11
	5186	2810	0.55	68.3%	100%	4.81	163.16
	5090	3015	0.46	68.55%	100%	4.87	51.13
MFCC	7950	3900	14.9	68.55%	100%	7.58	212.51
	7534	3745	10.69	68.35%	100%	6.82	365.16
	7965	3850	13.47	67.75%	100%	7.58	215.64

Tabela 5 – Three systems based on each type of descriptor. Results were averaged on the cross-validation result of 10 different combination of folds. SVMs were used as a classifier and applied on files (average of all frame descriptors).

Although not achieving the same performance we note that the best system for each descriptor using only one cross-validation estimate was also the best when using ten different combinations of folds (comparing to the mean and median of the best 10).

Finally, for its relative good results, fast computation and simplicity we choose to use log-spectrograms for most further evaluation of classifiers.

9.2 Classification based on frames

In this section we present results of a more complex way of classifying a large excerpt of audio. Although the descriptors and classifiers are inherently the same, the metodology is very different.

Here, we do not take the average of frame descriptors to classify a file, but we classify each frame to one of the classes. This is an approach inherited by the automatic speech recognition domain where HMMs are stacked on top of a regressor giving the system the ability to change from one class to another with time.

In pure classification, where we don't need any dynamics in the system, the regressor step is usually replaced by a classifier and the HMMs are replaced by a voting procedure, see Section 6.3.

9.2.1 Support vector machines

To apply this strategy to SVMs it was necessary for computational reasons to subsample the training set. Especially when using a high overlap between frames, the number of training examples becomes overwhelmingly high. We tried at first to use random

subsampling but this means outright discarding relevant data. Instead, we define a fixed number of examples per file, noted as n_e. We divide the frames of each file in n_e portions and we compute the average of each portion.

Similarly to Section 9.1.2, we start by tuning an SVM using the TPE algorithm and optimizing the parameters w_{len}, h_{len} and C. The prior distribution for each parameter was the same as presented in Section 9.1.2, expressions 9.4. Since we are using SVMs, the voting strategy was not weighted, i.e. the class of a file is the class where the greatest number of frames of this file were classified.

Results for this strategy applied on the mel-log-spectrum are displayed in Table 6.

	1	2	3	4	5	6	7	8	9	10
w_{len}	3350	5570	4180	4590	4310	6720	2390	5830	8410	7080
h_{len}	130	1010	4890	850	570	60	4080	2880	3680	2200
C	0.62	0.61	1.56	1.22	0.57	0.55	0.20	1.09	1.33	1.68
acc_v	78%	76%	75%	75%	75%	75%	74%	74%	74%	74%
acc_t	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
t_{fe}(s)	49.18	49.59	1.82	8.05	53.72	169.77	3.08	4.47	4.79	6.51
t_{tv}(min)	11.96	2.82	0.50	2.68	3.68	60.61	0.36	1.17	1.57	1.95

Tabela 6 – 10 best systems (out of 319) based on the log-spectrum, using an SVM and classifying by frames

	Best	Mean	Median
w_{len}	3350	5243	5080
h_{len}	130	2035	1605
C	0.62	0.94	0.85
acc_v	71.7%	69.3%	69.1%
acc_t	100%	100%	100%
t_{fe}(s)	43.91	7.29	8.89
t_{tv}(min)	123.72	15.42	18.97

Tabela 7 – Results for repeat (10 times) cross-validation on 3 systems based on the log-spectrum, using an SVM and classifying by frames

9.2.2 Classification based on the average of frames

Giving continuity to the approach described in the last section we present here the results from different classifiers taking as a classification instance the average of all frame descriptors, notably a logistic regressor (Section 6.2.1.1) and an MLP (Section 6.2.1.2)

9.2.2.1 Logistic regression

A logistic regressor has more parameters to tune than an SVM. Although the only mandatory parameter is the learning rate of the SGD (noted here λ), it is beneficial to

add regularization properties of the classifier and also alter the funcioning of the SGD algorithm.

Regularizing a logistic regressor can be made by altering the cost function, transforming it into an elastic-net logistic regressor. The parameters here are the ℓ_1 and ℓ_2 regularization parameters. A second option is to apply dropout on the inputs. The probability of dropout for each update is noted p_{drop}.

There are also some degrees of liberty to the optimization algorithm besides the learning rate λ, notably the size of the batches used to compute the gradient at each iteration and the momentum parameter α. Because of the low number of training examples at each round (80), we applied batch gradient descent instead of SGD.

The feature extraction parameters w_{len} and h_{len} were fixed to the ones obtained with SVMs.

9.2.2.1.1 Log-spectrum

To evaluate the logistic regressor applied on the log-spectrum we have fixed the following parameters:

$$\begin{aligned} w_{len} &= 2410 \\ h_{len} &= 120 \\ n_{epochs} &= 1000 \end{aligned}$$

And we optimize the other ones using TPE. The prior distribution used for each of these parameters was:

$$\begin{aligned} \ell_1, \ell_2 \in \mathbb{R} &\sim \ln\mathcal{U}(10^{-8}, 0.1) \\ p_{drop} \in \mathbb{R} &\sim \mathcal{U}(0, 1) \\ \lambda \in \mathbb{R} &\sim \ln\mathcal{U}(10^{-4}, 1) \\ \alpha \in \mathbb{R} &\sim \mathcal{U}(0, 1) \end{aligned}$$

	1	2	3	4	5	6	7	8	9	10
ℓ_1	7.7e-4	3.9e-4	2.0e-6	8.3e-7	4.6e-4	2.0e-4	2.2e-8	5.8e-4	9.1e-6	1.7e-3
ℓ_2	1.6e-4	1.6e-4	2.8e-6	5.5e-6	3.7e-5	9.4e-6	5.5e-5	1.6e-4	6.0e-5	3.7e-5
p_{drop}	0.46	0.77	0.51	0.41	0.72	0.31	0.22	0.39	0.50	0.49
λ	0.027	0.038	0.037	0.033	0.017	0.014	0.005	0.061	0.027	0.022
α	0.20	0.20	0.77	0.54	0.22	0.26	1.00	0.05	0.16	0.50
acc_v	72%	72%	71%	70%	70%	69%	68%	67%	67%	67%
acc_t	99.75%	100%	100%	100%	99.5%	98.5%	99%	99.75%	100%	99.5%
t_c(s)	11.72	11.65	13.35	12.28	11.90	12.03	13.48	12.28	12.49	11.81
t_{tv}(s)	8.81	8.83	8.90	8.82	8.83	9.03	8.90	8.80	8.80	8.82

Tabela 8 – 10 best systems (out of 251) based on the log-spectrum, using a logistic regressor and classifying by files (average of all frame descriptors)

	Best	Mean	Median
ℓ_1	7.7e-4	4.1e-4	3.0e-4
ℓ_2	1.6e-4	6.8e-5	4.6e-5
p_{drop}	0.458	0.479	0.472
λ	0.027	0.028	0.027
α	0.203	0.390	0.243
acc_v	63.7%	63.1%	64.4%
acc_t	90.475%	92.65%	90.2%
t_c(s)	3.27	3.72	3.62
t_{tv}(s)	76.91	76.30	78.09

Tabela 9 – Results for repeated (10 times) cross-validation on 3 systems based on the log-spectrum, using a logistic regressor and classifying by files (average of all frame descriptors)

9.2.2.1.2 Mel-log-spectrum

To evaluate the logistic regressor applied on the mel-log-spectrum we have fixed the following parameters:

$$w_{len} = 4580$$
$$h_{len} = 3220$$
$$n_{epochs} = 1000$$

And we optimize the other ones using TPE. The prior distribution used for each of these parameters was the same as in Section 9.2.2.1.2.

	1	2	3	4	5	6	7	8	9	10
ℓ_1	2.9e-5	7.2e-6	1.5e-6	2.2e-8	5.5e-6	2.0e-6	7.0e-7	6.3e-7	6.2e-5	1.2e-6
ℓ_2	3.3e-3	1.7e-7	4.7e-3	1.1e-7	2.4e-7	2.8e-5	9.8e-3	1.1e-4	2.1e-6	4.2e-5
p_{drop}	0.35	0.70	0.43	0.24	0.70	0.04	0.29	0.02	0.62	0.21
λ	0.060	0.102	0.048	0.001	0.143	0.009	0.007	0.011	0.185	0.017
α	0.57	0.32	0.06	0.87	0.48	0.30	0.02	0.38	0.26	0.36
acc_v	69%	68%	67%	66%	66%	66%	66%	66%	65%	65%
acc_t	91%	95.25%	96%	90.75%	95.75%	99%	86%	99.75%	85.5%	99.75%
t(min)	1.88	1.84	1.73	0.69	1.77	1.80	1.80	1.81	1.74	1.86

Tabela 10 – 10 best systems (out of 100) based on the mel-log-spectrum, using a logistic regressor and classifying by files (average of all frame descriptors)

	Best	Mean	Median
ℓ_1	2.9e-5	1.1e-5	1.7e-6
ℓ_2	3.3e-3	1.8e-3	3.5e-5
p_{drop}	0.352	0.361	0.323
λ	0.060	0.058	0.032
α	0.572	0.360	0.339
acc_v	66.5%	66.4%	65.2%
acc_t	99.825%	100%	100%
t_c(s)	3.75	6.56	3.80
t_{tv}(min)	1.93	1.96	1.93

Tabela 11 – Results for repeated (10 times) cross-validation on 3 systems based on the mel-log-spectrum, using a logistic regressor and classifying by files (average of all frame descriptors)

9.2.2.2 Multi-layer perceptron

To evaluate an MLP of one hidden layer applied on the mel-log-spectrum we have fixed the following parameters:

$$w_{len} = 4580$$
$$h_{len} = 3220$$
$$n_{epochs} = 1000$$

Similarly to the logistic regressor we use batch gradient descent because of the low number of training examples.

We will note here $\ell_{1,H}$ and $\ell_{2,H}$ the regularization parameter for the first (and only) hidden layer of this network, and $\ell_{1,L}$ and $\ell_{2,L}$ the parameters of the output layer with a softmax activation function (equivalent to a stacked logistic regressor). Dropout was only applied between the hidden layer and the output layer with probability p_{drop}. The number of hidden units is noted as n_u.

Again, we optimize the parameters that were not fixed using TPE. The prior distribution used for each of these parameters was:

$$\begin{aligned} n_u \in \mathbb{N} &\sim \mathcal{U}(3, 1000) \\ \ell_{1,H}, \ell_{1,L}, \ell_{2,H}, \ell_{2,L} \in \mathbb{R} &\sim \ln\mathcal{U}(10^{-8}, 0.1) \\ p_{drop} \in \mathbb{R} &\sim \mathcal{U}(0, 1) \\ \lambda \in \mathbb{R} &\sim \ln\mathcal{U}(10^{-4}, 1) \\ \alpha \in \mathbb{R} &\sim \mathcal{U}(0, 1) \end{aligned}$$

Results are displayed in Table 12.

	1	2	3	4	5	6	7	8	9	10
n_u	677	878	696	844	804	870	883	889	799	738
$\ell_{1,H}$	2.8e-6	5.4e-5	2.0e-5	7.1e-5	1.3e-4	9.3e-5	3.9e-5	4.6e-4	1.1e-4	2.6e-6
$\ell_{2,H}$	1.2e-5	2.9e-5	1.4e-4	2.7e-3	1.1e-4	2.0e-3	2.8e-3	4.8e-3	1.5e-3	6.3e-3
$\ell_{1,L}$	1.6e-3	6.1e-5	5.1e-5	6.3e-6	4.9e-4	1.7e-5	7.2e-6	5.4e-5	1.4e-6	1.5e-7
$\ell_{2,L}$	5.1e-6	8.2e-7	7.1e-7	2.3e-8	3.0e-7	1.4e-7	1.9e-8	1.7e-7	1.1e-7	7.5e-8
p_{drop}	0.080	0.040	0.020	0.018	0.040	0.086	0.171	0.066	0.050	0.055
λ	0.011	0.012	0.016	0.008	0.017	0.008	0.013	0.005	0.009	0.006
α	0.679	0.453	0.474	0.329	0.319	0.407	0.361	0.425	0.479	0.629
acc_v	72%	72%	71%	71%	71%	71%	71%	70%	70%	70%
acc_t	100%	100%	100%	100%	100%	100%	100%	99.5%	100%	100%
t(min)	2.48	2.58	2.06	3.02	3.13	3.31	3.34	1.93	2.09	2.11

Tabela 12 – 10 best systems (out of 1574) based on the mel-log-spectrum, using an MLP and classifying by files (average of all frame descriptors)

	Best	Mean	Median
n_u	677	808	824
$\ell_{1,H}$	2.8e-6	9.9e-5	6.2e-5
$\ell_{2,H}$	1.2e-5	2.0e-3	1.7e-3
$\ell_{1,L}$	1.6e-3	2.3e-4	3.4e-5
$\ell_{2,L}$	5.1e-6	7.4e-7	1.6e-7
p_{drop}	0.080	0.063	0.053
λ	0.011	0.010	0.010
α	0.679	0.455	0.439
acc_v	66%	64.5%	63.5%
acc_t	100%	100%	100%
t_c(s)	4.67	4.96	4.69
t_{tv}(min)	6.31	7.03	6.97

Given that these results are not considerably better than those obtained with an optimal logistic regressor and that we have enough capacity to attain 100% accuracy in the training set we choose not to go beyond one hidden layer.

9.3 Convolutional Neural Networks

In this section we present an approach that is most likely the "deeper" often used in audio classification. This is because we don't resort to averaging the frames of a long signal to obtain a feasible feature dimension as in Section 9.2.2 and we don't use any kind of voting to classify a long signal based on the classes assigned to its frames as in Section 9.2.

Still, unlike in image processing, the network does not take as an input the raw audio data. It relies on a simple transformation such as the ones described before. Each frame is transformed to a frequency representation of the data and a long signal is represented as a whole by a matrix, often a spectrogram.

For this experiments we have used the log-spectrogram of a signal as the descriptor for an individual instance. Here, the different frequency bins are not treated as a dimension of the CNN, but as different channels.

We have fixed the overall structure of the network: three convolutional layers and a fully-connected layer from the concatenation of outputs of the last convolutional layer.

For each convolutional layer indexed by i we optimize the width of the filters (Δ_i), the number of filters ($n_{k,i}$), its input dropout rate ($p_{drop,i}$), the regularization parameters ($\ell_{1,Hi}$ and $\ell_{2,Hi}$) and the max-pooling ratio of its output (ϕ_i). This totalizes 6 parameters for each convolutional layer.

The fully-connected layer also has a dropout rate ($p_{drop,L}$) and regularization parameters ($\ell_{1,L}$ and $\ell_{2,L}$).

Besides the parameters concerning each individual layer, we also have the SGD parameters as usual, the learning rate λ and momentum parameter α.

The prior sampling distribution was set as following:

$$
\begin{aligned}
\log(w_{len}) \in \mathbb{N} &\sim \mathcal{U}(8, 12) \\
\log(h_{len}) \in \mathbb{N} &\sim \mathcal{U}(6, 11) \\
n_{k,0}, n_{k,1} \in \mathbb{N} &\sim \mathcal{U}(2, 50) \\
n_{k,2} \in \mathbb{N} &\sim \mathcal{U}(2, 30) \\
\Delta_0, \Delta_1, \Delta_2 \in \mathbb{N} &\sim \mathcal{U}(2, 5) \\
\phi_0, \phi_1, \phi_2 \in \mathbb{N} &\sim \mathcal{U}(2, 6) \\
\ell_{1,H0}, \ell_{1,H1}, \ell_{1,H2}, \ell_{1,L} \in \mathbb{R} &\sim \ln\mathcal{U}(10^{-8}, 0.1) \\
\ell_{2,H0}, \ell_{2,H1}, \ell_{2,H2}, \ell_{2,L} \in \mathbb{R} &\sim \ln\mathcal{U}(10^{-8}, 0.1) \\
p_{drop,H0}, p_{drop,H1}, p_{drop,H2}, p_{drop,L} \in \mathbb{R} &\sim \mathcal{U}(0, 1) \\
\lambda \in \mathbb{R} &\sim \ln\mathcal{U}(10^{-8}, 1)
\end{aligned}
$$

$$\alpha \in \mathbb{R} \sim \mathcal{U}(0,1)$$

Results are displayed in Table 13.

	1	2	3	4	5	6	7	8	9	10
w_{len}	9	8	9	10	8	9	9	11	8	9
h_{len}	10	9	10	11	10	11	11	9	10	10
$p_{drop,0}$	0.079	0.213	0.072	0.117	0.037	0.085	0.081	0.280	0.310	0.031
$n_{k,0}$	50	49	22	28	42	33	41	25	21	46
Δ_0	4	3	5	4	2	4	4	4	2	3
$\ell_{1,h0}$	2.5e-8	4.8e-8	1.5e-8	8.2e-4	1.4e-6	6.5e-4	8.3e-5	3.7e-6	7.4e-7	1.9e-7
$\ell_{2,h0}$	3.5e-5	2.3e-5	9.4e-4	4.9e-7	2.1e-5	5.8e-7	4.7e-6	2.3e-6	9.6e-4	6.5e-5
ϕ_0	3	4	3	2	4	2	2	2	2	4
$p_{drop,1}$	0.103	0.024	0.119	0.120	0.174	0.139	0.175	0.222	0.136	0.142
$n_{k,1}$	29	30	23	14	26	16	22	28	27	20
Δ_1	5	5	4	3	4	2	3	4	4	5
$\ell_{1,h1}$	9.0e-6	3.4e-6	6.6e-6	5.9e-8	4.4e-6	1.1e-7	6.2e-7	1.3e-5	2.9e-5	6.6e-4
$\ell_{2,h1}$	9.6e-5	8.7e-7	2.6e-4	9.5e-4	1.8e-6	5.3e-4	7.5e-4	1.5e-4	5.2e-8	2.4e-6
ϕ_1	4	6	2	3	6	4	4	5	7	6
$p_{drop,2}$	0.267	0.399	0.357	0.399	0.142	0.093	0.123	0.400	0.287	0.186
$n_{k,2}$	19	25	19	17	20	17	16	8	17	15
Δ_2	5	5	5	4	5	5	5	5	4	5
$\ell_{1,h2}$	2.2e-6	1.4e-6	2.4e-7	5.5e-8	1.3e-5	1.0e-8	8.0e-8	8.1e-8	6.5e-8	4.3e-8
$\ell_{2,h0}$	2.4e-5	1.2e-4	2.7e-4	2.7e-4	5.4e-5	4.2e-4	9.9e-4	2.9e-4	4.4e-4	4.5e-6
ϕ_2	3	3	3	3	3	2	3	3	2	3
$p_{drop,L}$	0.078	0.275	0.096	0.007	0.143	0.061	0.135	0.001	0.165	0.058
$\ell_{1,L}$	1.5e-6	5.9e-8	1.3e-7	9.5e-8	5.2e-8	1.5e-7	1.7e-7	1.4e-5	3.1e-7	1.2e-6
$\ell_{2,L}$	1.1e-6	5.0e-5	2.1e-6	2.5e-7	5.8e-5	5.7e-7	2.8e-6	8.4e-6	2.8e-5	5.9e-5
λ	0.004	0.011	0.008	0.001	0.022	0.002	0.004	0.004	0.006	0.002
α	0.485	0.518	0.475	0.903	0.470	0.720	0.610	0.542	0.580	0.634
acc_v	62%	59%	56%	56%	55%	55%	55%	55%	54%	54%
acc_t	100%	99.25%	99.75%	96%	99.5%	100%	100%	100%	92.25%	98.25%
t_{fe}(s)	1.01	1.07	1.04	1.05	0.73	0.78	0.78	4.96	0.68	0.97
t_{cp}(s)	22.33	36.29	25.24	29.39	20.48	30.35	27.42	40.85	19.29	21.56
t_{tv}(min)	4.97	2.46	3.28	3.96	1.60	2.63	2.79	15.44	1.44	3.69

Tabela 13 – 10 best systems (out of 193) based on the log-spectrum, using a CNN

9.3.1 Undocumented results

The list of systems tested in this work presented in the previous sections eventhough long is not exhaustive. Structuring the training and validation procedure and carefully logging results to allow full interpretation weeks after the experiment is a task that demands a lot of work and when experimenting and prototyping different systems this is sometimes not convenient.

Some observations are worthy of mention.

9.3.1.0.1 Aggregated frame descriptors

First, a lot of time was spent optimizing not the size of the frames, but the number of adjacent frames to form an aggregated medium-term feature, especially using MFCCs of reduced size (12) plus the energy of the frame. No significant improvement was obtained from multiple to one individual frame as a classification instance.

9.3.1.0.2 Small dimension inputs

When performing classification with low-dimensional inputs, MLPs with two to three layers were more successful than a simple logistic regressor or a shallow network (only one layer). The dimensionality of the inputs is directly linked to the capacity of these models to be able to linearly classify instances. The smaller is the dimension, more non-linearities are needed. This is especially obvious in logistic regressors, where the number of parameters is proportional to the size of the input.

9.3.1.0.3 Unsupervised feature learning

Unsupervised feature learning was applied with the UrbanSound database over the aggregated MFCC vectors. Pre-training layers with RBMs and DBNs and fixing the parameters of these layers was yielding similar performance to training the whole network in a supervised way, as well as initializing certain layers with values obtained with pre-training.

We also observed that the particular implementation used for training RBMs was very slow in comparison to supervised training, for reasons to be investigated.

A number of possibilities could be the reason why unsupervised training didn't yield better results:

1. The UrbanSound database even if big is not varied enough to also capture all characteristics present in our smaller database.

2. The parameter space was possibly not explored well enough, since the time necessary to pre-train one model was considerably high (in the order of hours).

3. The role of the first layers of the network could be not very important for this task, being useful only to project a low-dimensional space into a higher one.

10 Conclusion and future work

This internship served as a review and an entry point for research in Audio Classification at Technicolor. A complete workflow was implemented and a large variety of methods was tested, hoping to evaluate the potential of different features (e.g. Spectrograms, MFCCs), classification strategies (e.g. average of frames, bag-of-frames, feature aggregation), classification methods (e.g. SVMs, logistic regressors, MLPs) and general approaches (pure supervised learning or integration of the feature learning step in the workflow).

The framework of deep learning proved itself usable, although not ideal for this problem setup. Most likely the reason for this was the quantity of labeled data, since deep learning has been growing to be the predominant approach in speech recognition, a very similar task where the availability of labeled data is higher.

Even if at an early stage, the research activity in this domain in Technicolor already starts to see by-products at the technology licesing side: two invention disclosures were submitted, both results from the internship. The first related to an unsupervised feature learning strategy and the second related to optimization algorithms for training general networks.

The main idea for future work would be to include external training examples in the learning process that are not part of the database used to evaluate results.

Another lead would be to follow further on the semi-supervised learning strategy. The lack of reliable and easy-to-use implementations of CDBNs imposed some difficulties in the internship since this was the best shot for efficient deep unsupervised feature learning for audio.

Referências

1 IMAI, S. Cepstral analysis synthesis on the mel frequency scale. In: IEEE. *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP'83.* [S.l.], 1983. v. 8, p. 93–96. Citado na página 21.

2 GIANNOULIS, D. et al. Detection and classification of acoustic scenes and events. *an IEEE AASP Challenge*, 2013. Citado 2 vezes nas páginas 21 e 57.

3 CORTES, C.; VAPNIK, V. Support-vector networks. *Machine learning*, Springer, v. 20, n. 3, p. 273–297, 1995. Citado na página 21.

4 REYNOLDS, D.; ROSE, R. C. et al. Robust text-independent speaker identification using gaussian mixture speaker models. *Speech and Audio Processing, IEEE Transactions on*, IEEE, v. 3, n. 1, p. 72–83, 1995. Citado na página 21.

5 BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, IEEE, v. 35, n. 8, p. 1798–1828, 2013. Citado 2 vezes nas páginas 21 e 31.

6 HUMPHREY, E. J.; BELLO, J. P.; LECUN, Y. Moving beyond feature design: Deep architectures and automatic feature learning in music informatics. In: CITESEER. *ISMIR*. [S.l.], 2012. p. 403–408. Citado na página 22.

7 MOHAMED, A.-r.; DAHL, G. E.; HINTON, G. Acoustic modeling using deep belief networks. *Audio, Speech, and Language Processing, IEEE Transactions on*, IEEE, v. 20, n. 1, p. 14–22, 2012. Citado na página 23.

8 MOHAMED, A.-r. et al. Deep belief networks using discriminative features for phone recognition. In: IEEE. *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*. [S.l.], 2011. p. 5060–5063. Citado na página 23.

9 LEE, H. et al. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: ACM. *Proceedings of the 26th Annual International Conference on Machine Learning*. [S.l.], 2009. p. 609–616. Citado 2 vezes nas páginas 24 e 34.

10 BENGIO, Y. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, Now Publishers Inc., v. 2, n. 1, p. 1–127, 2009. Citado 2 vezes nas páginas 24 e 31.

11 AUCOUTURIER, J.-J.; DEFREVILLE, B.; PACHET, F. The bag-of-frames approach to audio pattern recognition: A sufficient model for urban soundscapes but not for polyphonic music. *The Journal of the Acoustical Society of America*, Acoustical Society of America, v. 122, n. 2, p. 881–891, 2007. Citado na página 25.

12 HINTON, G. et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, IEEE, v. 29, n. 6, p. 82–97, 2012. Citado na página 26.

13 PEARSON, K. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Taylor & Francis, v. 2, n. 11, p. 559–572, 1901. Citado na página 29.

14 LEE, H. et al. Unsupervised feature learning for audio classification using convolutional deep belief networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2009. p. 1096–1104. Citado 2 vezes nas páginas 29 e 43.

15 BOGERT, B. P.; HEALY, M. J.; TUKEY, J. W. The quefrency alanysis of time series for echoes: Cepstrum, pseudo-autocovariance, cross-cepstrum and saphe cracking. In: CHAPTER. *Proceedings of the symposium on time series analysis*. [S.l.], 1963. v. 15, p. 209–243. Citado na página 29.

16 RAO, K. R.; YIP, P. *Discrete cosine transform: algorithms, advantages, applications*. [S.l.]: Academic press, 2014. Citado na página 29.

17 SMOLENSKY, P. Information processing in dynamical systems: Foundations of harmony theory. Department of Computer Science, University of Colorado, Boulder, 1986. Citado na página 33.

18 HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation*, MIT Press, v. 18, n. 7, p. 1527–1554, 2006. Citado 2 vezes nas páginas 33 e 35.

19 WELLING, M.; ROSEN-ZVI, M.; HINTON, G. E. Exponential family harmoniums with an application to information retrieval. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2004. p. 1481–1488. Citado na página 34.

20 WANG, N.; MELCHIOR, J.; WISKOTT, L. An analysis of gaussian-binary restricted boltzmann machines for natural images. In: *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)*. [S.l.: s.n.], 2012. p. 287–292. Citado na página 34.

21 NOROUZI, M.; RANJBAR, M.; MORI, G. Stacks of convolutional restricted boltzmann machines for shift-invariant feature learning. In: IEEE. *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. [S.l.], 2009. p. 2735–2742. Citado na página 34.

22 BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, IEEE, v. 5, n. 2, p. 157–166, 1994. Citado na página 35.

23 RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Cognitive modeling*, v. 5, p. 3, 1988. Citado 2 vezes nas páginas 35 e 40.

24 HINTON, G. E. Training products of experts by minimizing contrastive divergence. *Neural computation*, MIT Press, v. 14, n. 8, p. 1771–1800, 2002. Citado na página 35.

25 BENGIO, Y. et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, MIT; 1998, v. 19, p. 153, 2007. Citado na página 35.

26 BURGES, C. J. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, Springer, v. 2, n. 2, p. 121–167, 1998. Citado na página 37.

27 COX, D. R. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, JSTOR, p. 215–242, 1958. Citado na página 39.

28 HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks*, Elsevier, v. 2, n. 5, p. 359–366, 1989. Citado na página 40.

29 LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, IEEE, v. 86, n. 11, p. 2278–2324, 1998. Citado na página 41.

30 ABDEL-HAMID, O. et al. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In: IEEE. *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*. [S.l.], 2012. p. 4277–4280. Citado na página 44.

31 HINTON, G. E. et al. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. Citado na página 45.

32 DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, JMLR. org, v. 12, p. 2121–2159, 2011. Citado na página 48.

33 LECUN, Y. A. et al. Efficient backprop. In: *Neural networks: Tricks of the trade*. [S.l.]: Springer, 2012. p. 9–48. Citado na página 51.

34 SNOEK, J.; LAROCHELLE, H.; ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2012. p. 2951–2959. Citado na página 51.

35 RASMUSSEN, C. E. Gaussian processes for machine learning. Citeseer, 2006. Citado na página 52.

36 BERGSTRA, J. S. et al. Algorithms for hyper-parameter optimization. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2011. p. 2546–2554. Citado na página 52.

37 BERGSTRA, J. et al. Theano: a CPU and GPU math expression compiler. In: AUSTIN, TX. *Proceedings of the Python for scientific computing conference (SciPy)*. [S.l.], 2010. v. 4, p. 3. Citado na página 55.

38 COLLOBERT, R.; KAVUKCUOGLU, K.; FARABET, C. Torch7: A matlab-like environment for machine learning. In: *BigLearn, NIPS Workshop*. [S.l.: s.n.], 2011. Citado na página 55.

39 GIANNOULIS, D. et al. A database and challenge for acoustic scene classification and event detection. In: IEEE. *Signal Processing Conference (EUSIPCO), 2013 Proceedings of the 21st European*. [S.l.], 2013. p. 1–5. Citado na página 57.

40 SALAMON, J.; JACOBY, C.; BELLO, J. P. A dataset and taxonomy for urban sound research. In: ACM. *Proceedings of the ACM International Conference on Multimedia*. [S.l.], 2014. p. 1041–1044. Citado na página 57.