Detection and characterization of singing voice using deep neural networks

Jimena ROYO-LETELIER

Deezer Internship

ATIAM Master Program 2014-2015

Supervisors

Romain Hennequin

Manuel Moussallam

August 7, 2015

In this internship we investigate the feasibility of analyzing large audio database using deep neural networks. The goal is to implement an automatic recognition/classification system capable of analyzing large amounts of audio data in an industrial context.

This report is organized as follows: in the Chapter 1 we present the problematics of information retrieval from large audio database and we explain why music information retrieval methods, and in particular deep neural networks, seems to be an appropriated tool for this task. We also give a brief overview of previous works that have developed this approach. In Chapter 2 we present the general framework in machine learning and the problem of classification task using supervised learning. In Chapter 3 we present neural network models for supervised classification. In Chapter 4 we present our experiment using convolutional neural network for audio content classification. Finally, in Chapter 5 we discuss the results obtained and future work on this subject.

I would like to express my gratitude to all the people in the Deezer's Big Data and Research and Development teams for their sympathy and help during this internship. Specially to my supervisors R. Hennequin and M. Moussallam for their constant availability and support. It was a real pleasure to work with and learn from them. I will also like to thanks A. Hérault for his kindness and support in Deezer's company.

Contents

Chapter 1

Introduction

In this chapter we first describe the general principles of the music information retrieval field and deep systems. Then, we explain why they can be helpful for problems related to audio content at large scale. Finally, we give an overview of recent results on deep neural networks used in audio recognition/classification tasks.

1.1 Computer analysis of large audio databases

The field of computer audio analysis, and more specifically the subfield known as ***Music Information Retrieval (MIR)*** has seen dramatic changes over the past decade, benefiting from major advances in signal processing, machine learning and computer science, both theoretical and practical ones. MIR concepts, algorithms and tools are now widely recognized and used in both academic and industrial research contexts.

While a vast majority of MIR research is conducted on somehow synthetic datasets, remaining challenges include scaling up of MIR methods and designing solutions tailored for truly industrial applications.

Deezer is an international music streaming service, with a catalog rich of more than 35 millions of unique audio tracks serving dozens of millions of users on a daily basis in more than 180 countries. Finely characterizing this huge amount of data is a very challenging task. Metadata can be collected using internal and external databases, but those are filled with errors and only cover a fraction of the entire catalog.

In practice, Deezer audio content ranges from purely musical content of roughly any genre, style, period and quality, to speech contents such as audiobooks, interviews, political speeches and even to various kind of noises. Organizing and qualifying all of those is crucial for the service, and MIR (in complement whit other tools) seem to be the right solution for this. Indeed, from one side, human based extraction of information for large databases is quite expensive and may take very long time, and from another side, well implemented automatic methods extracting information directly from audio signals are far more reliable than other computational techniques.

In this internship we studied the feasibility of using deep neural networks for the automatic classification of Deezer's audio content. We addressed two classification problems (male vs female voice and speech vs music) and we have implemented an automatic system to solve these. Since the analysis of the whole catalog is expensive in calculation time and memory resources, we took

a special care in implementing fast and scalable algorithms. In particular, one of the reasons to use deep neural networks over others MIR classification systems is their fast inference phase, which allows for an efficient treatment of large amounts of data.

1.2 Music Information Retrieval and Deep Systems

General Framework

From a practical point of view MIR focuses on the development of computational systems to solve a wide range of music related tasks, such as automatic identification of tempo, source separation or genre classification.

The general framework for solving tasks in MIR may be schematized as follows (see Figure 1.1 for a graphic representation): using signal processing techniques, raw audio data is first transformed into a sparser representation such as the ***short-term Fourier transform (STFT)*** or the ***constant Q transform (CQT)***. From these representations audio ***descriptors*** (also called audio ***features***) are derived (for example Mel-Frequency Cepstral Coefficients or Chroma). These features are used as inputs to machine learning systems (for example support vector classifiers or random forests), which finally yield a solution for the relevant task.

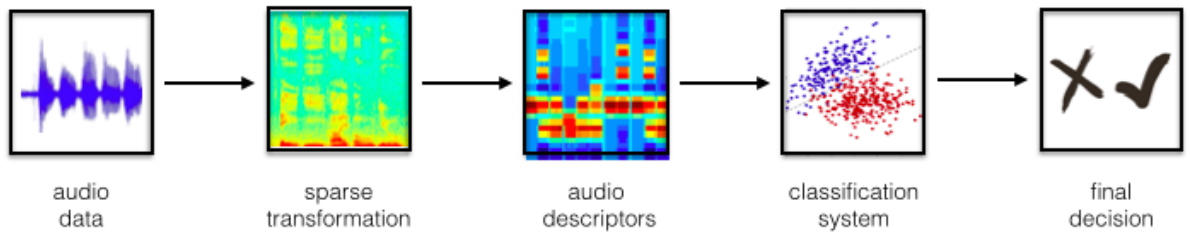


Figure 1.1: Schematic representation of the general framework in MIR. Here we show an example for an automatic audio classification task.

Deep systems

In machine learning jargon a system is called ***deep*** if it has a structure with several stages, in which the input data is sequentially transformed from one representation to another. As stated in [HBL12], each stage uses simpler elements to produce an "emergent whole greater than the sum of its parts". Robust music representations can thus be achieved in this way by breaking larger systems into a hierarchy of simpler parts.

An example of such a deep system is the tempo estimation method from [Sch98]. Indeed, in this procedure the input audio is processed by several stages involving linear (frequency bands decomposition, pooling, filtering) or non-linear (logarithmic scale, rectification) transformations. See Figure 1.2 for a scheme of this algorithm.

We mention that besides the examples before mentioned, another multi-scale system recently developed is the ***scattering transform*** [AM11, AM14]. This method extends classical audio descriptors by computing representations of multiple orders, through cascades of wavelet convolutions and modulus operators. It has been shown to capture long-term information for audio

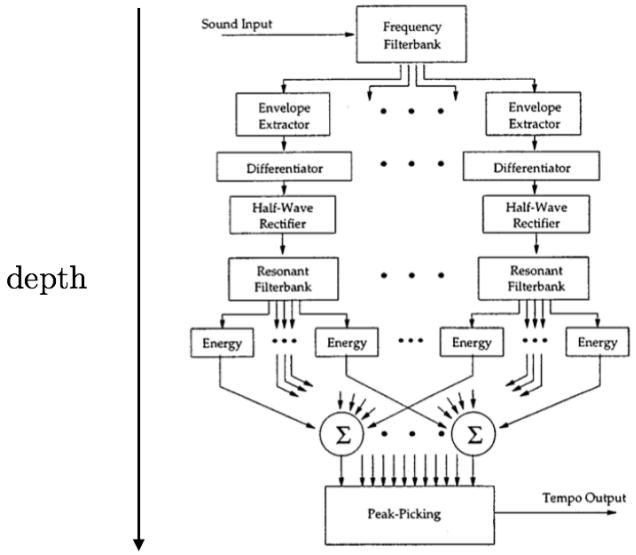


Figure 1.2: Scheme of the deep structure involved in a tempo estimation algorithm. Image from [Sch98].

classification [HBL12].

Automatic feature extraction

Audio descriptors are designed to reveal information about the input data that is relevant for a given task. They give information about temporal, spectral or energetic properties of the audio. There is a great variety of features available (see [Pee04] for a review), but ***Mel-Frequency Cepstral Coefficients (MFCC)*** and ***Chroma*** are the most widely used.

One of the main interests in using ***deep neural networks (DNNs)*** for audio content related task is to merge the feature extraction step and the purely machine learning one [HBL12]. This means that we provide to the system raw (such as audio signal) or sparse (such as spectrograms) data, and we let it learn the "most suitable" features to be used in the higher levels of the system. Since the selection of audio features requires a considerable expertise about the task to solve [DS14] and can be a complex and time-consuming process [DS13], the automatic extraction of audio features is currently a subject of interest in the MIR community.

In Figures 1.3 and 1.4 we show the results of two different experiment using DNNs (see Section 1.3 here below for more details). In the first one, a network was trained to perform onset detection using spectrograms as input data. We see that the system has learned a way to detect strong temporal variations of energy in the spectrograms. In the second experiment, another different network was trained to perform genre classification using audio signals as input data. We see that the system has learned features similar to sinusoidal signals. We conclude that –in this two examples DNNs– are indeed capable of automatically extracting meaningful low level features, that coincide with the ones that one could intuitively expect.

1.3 Deep Neural Networks applied to audio related task

Recent results show that deep neural networks may outperform previous state-of-the-art classification techniques in audio recognition/classification tasks, while requiring less manual preprocessing [HBL12]. Without being extensive, we present here some of these results and we refer to [HBL12] for complete references.

In [DS14] the authors investigate the use of raw audio signals instead of spectrograms as input to a system called ***Convolutional Deep Belief Network (CDBN)*** (see [Ben09]), and evaluate their performance in a genre classification task on the Magnatagatune dataset [LWM+09]. The performance is not improved with respect to MFCC input data, but the authors claim that the network learned features that correspond to frequency basis (sinusoids), as well as phase and translation invariant features, see Figure 1.4.

In [SB14] onset detection was performed using Convolutional Neural Networks. The network is trained with spectrograms playing the role of images in a vision recognition system. Visualizations of both the filter weights learnt by the model and the activations of its unit are provided, which greatly helps to understand the network behavior, see Figure 1.3.

In [HE10] a feature extraction system consisting of a deep belief network (DBN), combined with a non-linear support vector machine (SVM) classifier, is used to automatically discriminate between musical genres on the Tzanetakis dataset [TC01]. The learned features of the DBN compare favorably against MFCCs as input for the SVM classifier.

In [WWS11] automatic recognition of singer traits (gender, age, height and "race") is performed using Bidirectional Long Short-Term Memory recurrent neural networks on a extended version of the UltraStar database [SKW+10]. The authors implement source separation techniques to enhance harmonic parts and extract the leading voice as data preprocessing.

In [LPLN09] CDBNs are used to various audio classification tasks using unlabeled data. The authors claim that in the case of speech data (timit dataset [FDGM86]), the features learnt by the network correspond to phonemes. Sparsity regularization terms are added to the log-likelihood objective function to obtain sparse activations.

The major difference of our approach with the previous mentioned works is that we try to make use of the invariance properties of the CNN in conjunction with the structure of audio signals. We also addressed a different problem, knowing the speech vs music classification of audio content.

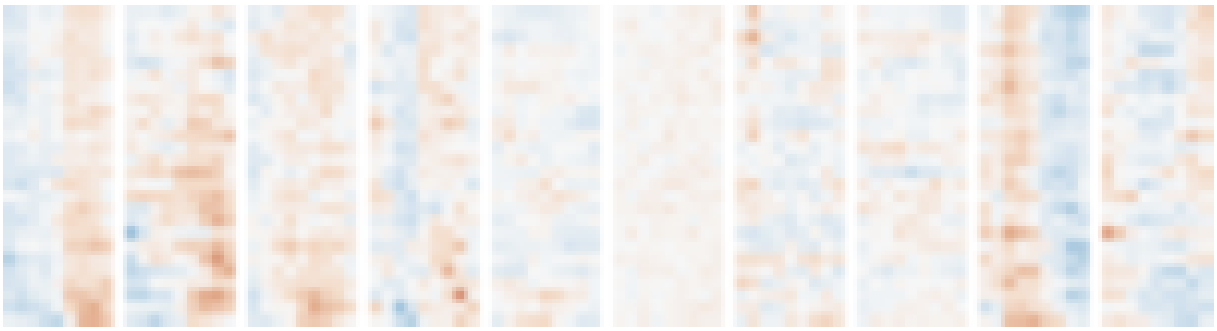
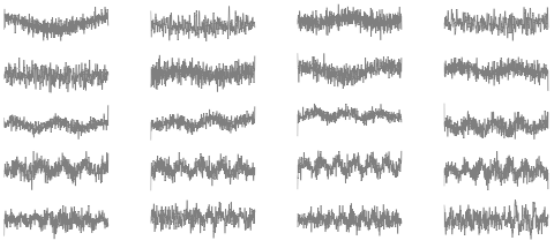
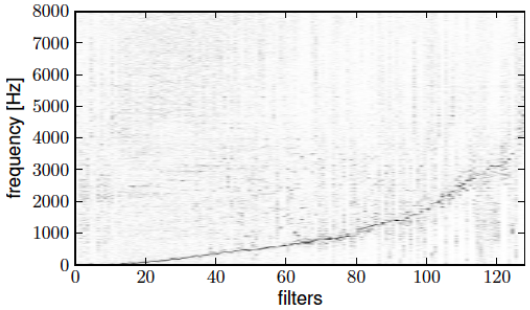


Figure 1.3: Filters learned at the penultimate layer unit of a CNN trained in [SB14] to perform onset detection. Time increases from left to right and frequency from bottom to top.



(a) Some of the filters learned.



(b) Normalized magnitude spectra of the filters learned ordered according to the dominant frequency (from low to high).

Figure 1.4: Filters learned by the lowest layer of a DBN trained in [DS14] to perform genre classification using directly with audio signals as input data.

Chapter 2

Machine Learning

2.1 Supervised Learning for Classification Problems

In this section we briefly present the general framework in machine learning techniques, particularly applied to classification tasks. We explain the concept of automatic classification, the problem of label data samples by associating a class to each of them, and the supervised learning procedure used to solve it.

2.1.1 Formal Statement of the Problem

The main goal in machine learning is to solve a task using a –statistical or computational– ***model*** that depends on some ***parameters*** $\theta \in \Theta$. The "learning" of the model refers then to the adjustment of these parameters in order to obtain the best possible performance on the required task. The learning can be "supervised" or "unsupervised", depending on whether we use or not a ***training dataset*** to guide the learning.

In ***classification*** tasks we have an ***input space*** $\mathcal{X}$ and a discrete number of M ***classes*** $y_m, m = 1, \ldots, M$. The task is to predict the class membership $y(x)$ of the samples $x \in \mathcal{X}$. The training data $X \times Y$ consists of a set of N samples $x^{(i)}, i = 1, \ldots, N$ for which the corresponding ***ground true*** classes $y^{(i)}$ are known.

The ***prediction model*** has two major components. First, a ***score function***:

$$h = h_\theta : \mathcal{X} \to \mathbb{R}^M \tag{2.1}$$

that depends on the parameters. The membership of a sample x to a class is determined by the value of $h(x)$. For example, if x belongs to the m-th class, then the value of the m-th coefficient of $h(x)$ will be high, while the other coefficients will have small values. We will see in Section 2.1.2 what this formally means in terms of probability distributions.

The second component of the model is a ***loss function***:

$$\mathcal{L} = \mathcal{L}_{X,Y} : \Theta \to \mathbb{R}\,,$$

that quantifies the agreement between the predicted classes and the ground truth classes. Notice that the loss is a function of the parameters but it depends on the training data.

The best possible performance, if it exists, is obtained with the parameters that solve the optimization problem:

$$\hat{\theta} = \text{argmin}_{\theta\in\Theta}\, \mathcal{L}_{X,Y}\,. \tag{2.2}$$

Usually, there is no simple analytic formulation for $\hat{\theta}$, so we try to approximate it with the limit of a sequence $\{\theta_n\}_n$ that –hopefully– decreases the value of the loss. How θ_{n+1} is obtained from θ_n is called the ***updating rules*** of the model.

Notice that besides θ, other numerical parameters may play a role in the choice of the score function or the loss. These are called ***hyper-parameters*** of the model and are not considered in the optimization problem (2.2).

2.1.2 Decision Regions and Probabilistic Interpretation

There are several ways of interpreting the score function h. In a first way, we partition the input space in ***decision regions*** defined by

$$\Omega_m = \{x \in \mathcal{X}\,;\, y(x) = y_m\}\,, \quad m = 1, \dots, M\,;$$

and we assume that each of these regions can be approximated by a region where h_m (the m-th coefficient of h) takes a range of values (for example Ω_0 would be similar to $\{x \in \mathcal{X}\,;\, h_0(x) \geq 0\}$). The learning of the model consists then in getting the best parameters θ in order to approximate these regions.

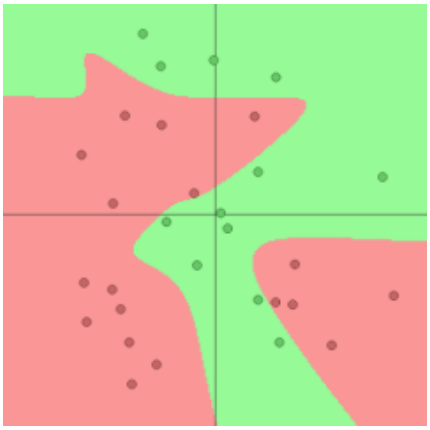


Figure 2.1: Cartoon representation of the decision regions for two labels of an input space of dimension 2. Image from [Kar].

We may define a –kind of– indicator function $f : \mathcal{X} \to \mathbb{R}^M$ such that for every $m = 1, \dots, M$,

$$f_m(x) = 1 \quad \text{if} \quad y(x) = y_m \quad \text{and} \quad 0 \quad \text{otherwise.}$$

Since this function gives the exact result for the classification problem, h should approximate f as good as possible. This ability of the model to approximate different classes of function is called its ***representational power***.

In another –more statistical– approach, we assume that the function score gives the ***conditional distribution probability*** for the membership to a class m, given the training data and the parameters of the model. This writes

$$h_m(x) = P(y^{(i)} = y_m | X; \theta)\,, \tag{2.3}$$

where h_m denotes the m-th coefficient of h. In order to obtain a well defined probability distribution, we impose that

$$\sum_{m=0}^{M-1} h_m(x) = 1 \quad \text{and} \quad h_m(x) \in [0,1]\,, \quad \forall x \in \mathcal{X}\,, \quad \forall m = 0, \dots, M-1\,.$$

2.2 Examples

We now present practical examples of the theoretical framework in machine learning. We first present the simplest linear model used for classification, the maximum likelihood estimation principle and the more complex softmax classifier.

2.2.1 A Simple Linear Model

Let's assume that the input space is $\mathcal{X} = \mathbb{R}^N$. A simple example of a score function is the linear mapping $h = h_{W,b} : \mathbb{R}^N \to \mathbb{R}^M$ such that

$$h(x) = Wx + b\,.$$

The parameters of this model are the ***weight*** matrix $W \in \mathcal{M}_{M,N}(\mathbb{R})$ and the ***bias*** term $b \in \mathbb{R}^M$.

The membership of a sample x to a class m is determined by the value of $h_m(x)$ being positive or not. Denoting by W_m the m-th row of W we have

$$h_m(x) = W_m^T x + b_m\,,$$

so $\{h_m(x) \geq 0\}$ is an affine hyper-plane $\mathcal{H}_m$ in $\mathbb{R}^N$. Hence, if the decision regions are well approximated by an hyper-plane, the linear model should succeed in giving good class predictions. Notice that when the W_m is modified, $\mathcal{H}_m$ rotates in different directions, while modifying the bias b_m translate $\mathcal{H}_m$. The update rules for the parameters correspond then to tuning W_m and b_m to get the best hyper-plane $\mathcal{H}_m$.

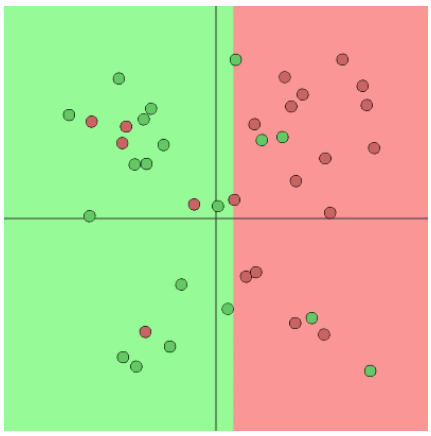


Figure 2.2: Separation regions given by a linear classifier for one class with an input space of dimension 2. Image from [Kar].

2.2.2 Log-likelihood

A general and extensively used principle for estimating the parameters of a statistical model is the ***maximum likelihood***. Suppose that the input data are independent and identically distributed random samples, which probability density is given by (2.3). We define the ***likelihood*** of the data given the parameters θ as the joint density function for all samples

$$L(\theta) = P(Y|X,\theta)\,.$$

Since the samples are i.i.d. we have

$$L(\theta) = \prod_{i=1}^{N} P(y^{(i)}|X,\theta)\,. \tag{2.4}$$

The principle of maximum likelihood assumes that the most reasonable values for θ are those for which the probability of each observed sample is maximal. Evidently, this is equivalent to maximizing any increasing function of $L(\theta)$. In particular, taking the log in (2.4) transform the product in to a sum, leading to more easy-to-handle expressions. The loss of the model is then defined as the ***negative log-likelihood*** of the data

$$\mathcal{L}_{X,Y}(\theta) = -\sum_{i=1}^{N} \log P(y^{(i)}|X,\theta)\,. \tag{2.5}$$

2.2.3 Logistic Regression and Softmax

The ***logistic regression*** is a non-linear model used in supervised classification with two classes problem. Similarly to the linear model exposed in Section (2.2.1), the parameters are a $1 \times N$ matrix W and a real value b. The score function is given by

$$h(x) = g(W^T x + b) \tag{2.6}$$

where g is the ***sigmoid*** (also called ***logistic***) function

$$g(t) = \frac{1}{1+e^{-t}}\,. \tag{2.7}$$

Notice that g is a smooth function which image is $[0,1]$ and

$$\begin{aligned} g(t) &\to 0 \qquad \text{as} \qquad t \to -\infty \\ g(t) &\to 1 \qquad \text{as} \qquad t \to +\infty\,. \end{aligned}$$

Any continuous function that satisfies the two conditions here before is called ***sigmoidal***.

Moreover, g have the nice property that

$$g'(t) = g(t)(1-g(t))\,. \tag{2.8}$$

This logistic regression model assumes that the conditional distribution of y given X and the parameters is given by

$$\begin{aligned} P(y=1|X,W,b) &= h(x) \\ P(y=0|X,W,b) &= 1-h(x) \end{aligned}$$

which also writes

$$P(y|X,W,b) = h(x)^y(1-h(x))^{(1-y)}\,.$$

The log-likelihood of this model as defined in (2.5) is then

$$\mathcal{L}_{X,Y}(\theta) = \sum_{i=1}^{N}\left[y^{(i)}\log h(x^{(i)}) + (1-y^{(i)})\log(1-h(x^{(i)}))\right]\,.$$

The ***softmax regression*** model is the generalization of logistic regression to an arbitrary number M of classes. The conditional distributions are assumed to be of the form

$$P(y=y_m|X,W,b) = \frac{e^{W_m^T x+b_m}}{\sum_{k=1}^{M} e^{W_k^T x+b_k}}\,, \tag{2.9}$$

so the log-likelihood writes

$$\mathcal{L}_{X,Y}(\theta) = \sum_{i=1}^{N}\sum_{k=1}^{M} 1_{\{y^{(i)}=y_k\}}\log\left(\frac{e^{W_k^T x^{(i)}+b_k}}{\sum_{j=1}^{M} e^{W_j^T x^{(i)}+b_j}}\right)\,. \tag{2.10}$$

We remark that softmax regression may be derived using the more general framework of Generalized Linear Models, see for example [JHTW13].

2.3 Training and overfitting

We now explain how a model is trained in practice and the different gradient descent methods used. We also give some ideas about the overfitting problem encountered when training a model and how to avoid it.

2.3.1 Gradient Descent Methods

When there is no analytic solution to the optimization problem in (2.2), the minimum may be found using ***gradient descent (GD)*** process, which uses the update rule

$$\theta_{n+1} = \theta_n - \eta\,\nabla_\theta\mathcal{L}_{X,Y}(\theta_n)\,. \tag{2.11}$$

The gradient of $\mathcal{L}_{X,Y}$ is associated with a vector in the parameter space Θ which points out in the direction in which $\mathcal{L}_{X,Y}$ growths the most. Hence, the update rule in (2.11) corresponds geometrically to move each parameter θ in a direction that lowers the value of the loss function by a step of η (see Figure 2.3). This later is a meta-parameter called the ***learning rate***.

Whenever this process converges depends of course on the form of the loss and the value of the learning rate. For a convex loss with a Lipschitz derivative, there is a small enough η that assures the convergence (see for example [All05]). Unfortunately, in many cases the loss function is highly non-convex. Many local minima may exist then and there is no warranties of convergence to a global minimum, making the training of the model difficult.

To accelerate computations, the gradient descent algorithm may be simplified by taking only one randomly chosen training sample $x^{(i)}$ at each iteration of the gradient descent process. This means that instead of minimizing the loss $\mathcal{L}_{X,Y}$ defined over the whole training dataset, we minimize a restricted loss $\mathcal{L}_{x^{(i)},y^{(i)}}$ where only one sample is considered. Hence, the update rule is

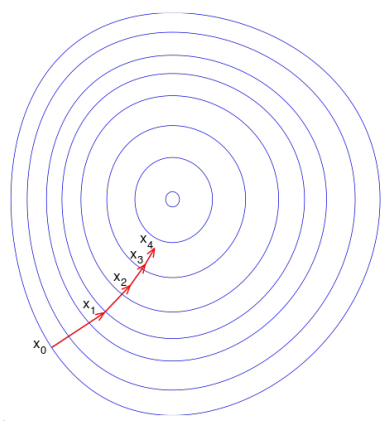


Figure 2.3: Gradient descent schematic representation. The drawn lines are the isolines of a regular function with a global minimum. Image from Wikipedia.

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathcal{L}_{x^{(i)},y^{(i)}}(\theta_n)\,. \tag{2.12}$$

This process is called ***stochastic gradient descent (SGD)*** or sometimes ***on-line gradient descent***. Whereas gradient descent must compute the derivatives of all the terms in the sum (2.5) before updating the parameters –a costly operation if the number of samples is large– stochastic gradient descent can modify the parameters (and decrease the value of the loss) before computing all the derivatives. Notice that since the loss is of the form

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \mathcal{L}_{x^{(i)},y^{(i)}}(\theta)\,,$$

GD corresponds with the mean of SGD on the whole training dataset.

The study of the convergence of stochastic gradient descent is out of the scope of this report, but it has been studied in the literature. There exists conditions on the learning rate and on the loss that assures convergence (see [Bot]).

Instead of considering only one sample, we may also take a partition $\{X_j\}_{j=1,\dots,J}$ of X and perform the update rules

$$\theta_{n+1} = \theta_n - \eta \frac{\partial \mathcal{L}_{X_j,Y_j}}{\partial \theta}(\theta_n)\,. \tag{2.13}$$

This is called ***mini-batch gradient descent (MSGD)***. The subsets X_j are called ***mini-batches*** and a whole pass by the J mini-batches is called an ***epoch***. As we will see later, this permits to parallelize –and thus speed– computations on the gradient descent process.

2.3.2 Generalization, Overfitting and Regularization

The ***overfitting*** of a model occurs when it makes very good predictions on the training data, but fails to give accurate predictions for new samples. This may be caused by training a complex model (in the sense that it has a large number of parameters) with too few training samples. Overfitting can be avoided by validating the model on a different dataset, hence called the ***validation*** dataset or by increasing the size of the train dataset. The ***generalization***

performance of a method refers to its capacity to make good predictions for independent *test* data. In practice, we split the available data for which the ground true classes $y^{(i)}$ are known in three sets: train, validation and test. Notice that to prevent overfitting and obtain a model that generalizes well, it is extremely important to have disjoint train, validation and test datasets.

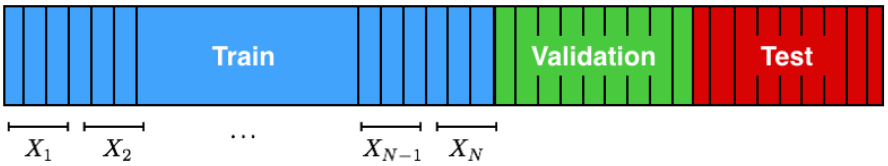


Figure 2.4: Splitting of the available data in train, validation and test datasets and its mini-batches.

In order to estimate the performance of the model during the training we measure how much ***errors*** it makes. This is done with the ***zero-one loss*** function

$$\mathcal{L}_{01}(\theta) = \sum_{i=1}^{N} \mathbf{1}_{\{y(x)=y^{(i)}\}} \tag{2.14}$$

computed over the validation dataset.

The final performance of the model is estimated over the test dataset, using the zero-one loss function or standard measures such as ***recall***, ***precision*** and ***F-measure***. If tp is the number of true positives, fp the number of false positives and fn the number of false negatives computed by the model, these are defined by

$$\text{recall} = \frac{tp}{tp+fp}\,, \quad \text{precision} = \frac{tp}{tp+fn} \quad \text{and} \quad \text{F-measure} = 2\,\frac{\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}}\,. \tag{2.15}$$

Intuitively, recall is the ability of the classifier to find all the positive samples while precision is the ability to not label as positive a sample that is negative. F-measure is a weighted average of the precision and recall.

Coming back to the overfitting problem, another way of preventing it is to restrict the magnitude of the values that the parameters can take. Indeed, this prevents the model from making too accurate predictions for the train dataset, which fails to predict new unseen examples. This is done by adding a ***regularization term*** to the loss. We thus minimize

$$\mathcal{L}_{X,Y,\lambda}(\theta) = \mathcal{L}_{X,Y}(\theta) + \lambda Q(\theta)\,, \tag{2.16}$$

where Q is usually a l_p***-norm*** on the space of parameters and λ is a meta-parameter called ***regularization strength*** which, as its name indicates, controls the strength of the regularization. When $\Theta \approx \mathbb{R}^K$ for example, the l_p-norm writes

$$\|\theta\|_p = \left(\sum_{k=1}^{K} |\theta_k|^p\right)^{1/p}$$

The most common regularizations use the ℓ_2 and ℓ_1 norms, or a combination of both. While ℓ_1-norm regularization leads to sparse parameters, where only a few have a "significant" value and the rest are very close to zero, ℓ_2-norm regularization leads to uniform "low" value parameters. The ***elastic net*** regularization use a linear combination of ℓ_1 and ℓ_2 norms. When the

parameters are double indexed, it is possible to use ***mixed norms*** regularization, which lead to parameters with "lines" of sparse coefficients (see [KT09]). When $\Theta \approx \mathbb{R}^K \times \mathbb{R}^L$ for example, the $l_{p,q}$-norm writes

$$\|\theta\|_{p,q} = \left(\sum_{\ell=1}^{L} \left(\sum_{k=1}^{K} |\theta_{k,\ell}|^p \right)^{p/q} \right)^{1/q} .$$

Another possibility is to use as regularization term the quotient between the ℓ_1 and ℓ_2 norms

$$Q(\theta) = \frac{\|\theta\|_1}{\|\theta\|_2} .$$

As with the ℓ_1-norm regularization, this leads to sparse parameters but the presence of the ℓ_2-norm in the denominator allows for less "peaky" coefficients.

Chapter 3

Neural Networks

We now present the neural network model for supervised classification. In the first part, we present the structure of a general feed-forward neural networks. In the second part, we try to give some insights about how these networks work.

3.1 Fully-Connected Neural Networks

3.1.1 Units, layers and non-linearities

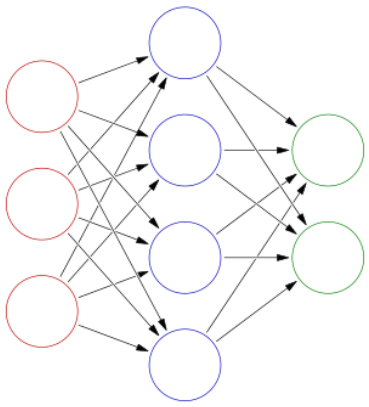


Figure 3.1: A feed-forward fully connected network. Image from Wikipedia.

As wee see in Figure 3.1, a ***neural network*** is a collection of connected ***units*** (also called ***neurons***) arranged in ***layers***. An unit take a value, called its ***activation***, which depends on the values of the other units to which it is connected and on the parameters of the network. The ***size*** of a layer is the number of units that it contains. In ***feedforward*** networks, the layers are hierarchically organized so the activations of an unit in a layer depends on the values of the units strictly below. The first and the last layers of a network are respectively called the ***input*** and ***output*** layers. The rest of the layer are designed as ***hidden***. The ***depth*** of a network refers to how many layers it contains (without counting the first input layer).

A common connectivity structure is the one of ***fully-connected*** networks (see Figure 3.1), in which units between two adjacent layers are fully pairwise connected, but units within a single layer share no connections. We see that in this case the information stored in the units "flows"

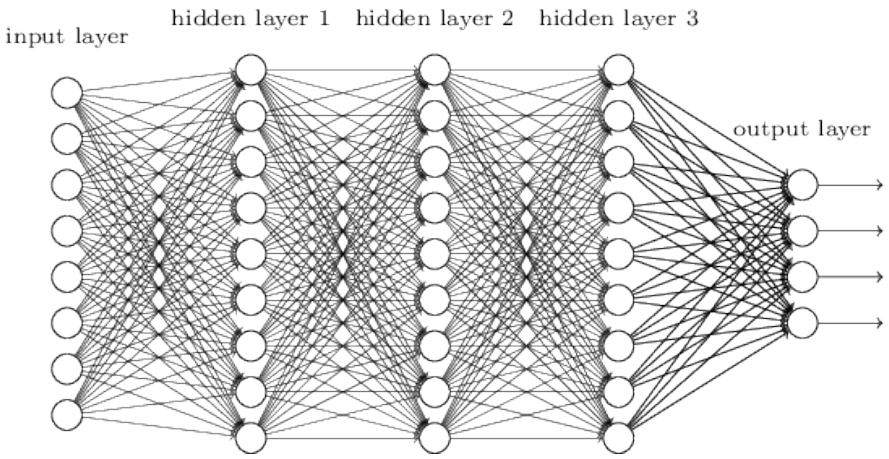


Figure 3.2: A fully connected feed-forward network with 3 hidden layers. Image from [Nie].

from lower to higher layers.

Consider a fully-connected network of depth L. The ℓ-th layer is represented by a vector $h^\ell \in \mathbb{R}^{M_\ell}$, where M_ℓ is it size. For $0 < \ell < L$ the parameters of the ℓ-th layer are a weight matrix $W^\ell \in \mathcal{M}_{M_\ell, M_{\ell-1}}(\mathbb{R})$ and a bias term $b^\ell \in \mathbb{R}^{M_\ell}$. The activation of its m-th unit is given by

$$h_m^\ell = \sigma\left((W_m^\ell)^T h^{\ell-1} + b_m^\ell\right), \tag{3.1}$$

where σ is a non-linear function called the ***non-linearity*** or ***activation function***. See Figure (3.3) for a graphic representation.

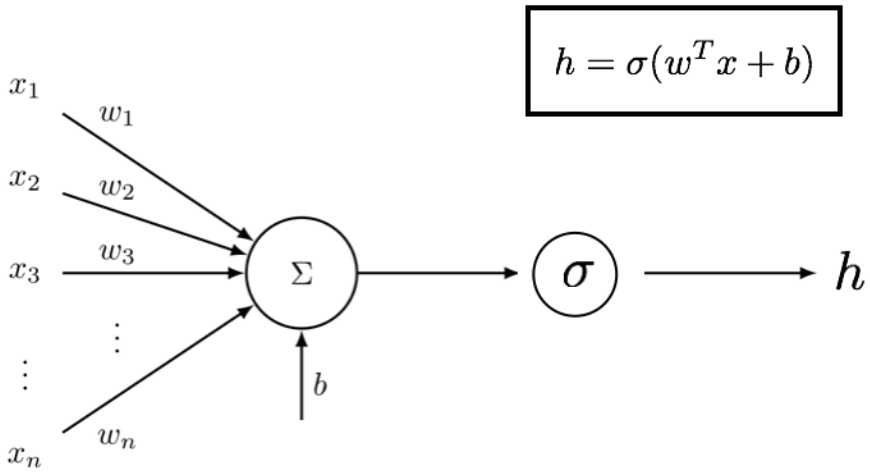


Figure 3.3: Activation of a single unit h in the first hidden layer of a neural network. Here $x = (x_1, \ldots, x_N)$ is the input vector, $w = (w_1, \ldots, w_N)$ a weight column and b the bias term.

When using a fully-connected network for a ***classification task***, the input layer contains the data samples that we want to classify, so $h^0 \in \mathcal{X}$. The output layer h^L is replaced by the softmax classifier detailed in Section 2.2.3, which is not a fully-connected layer. The value of its

m-th unit is given by

$$h_m^L = \frac{e^{(W_m^L)^T h^{L-1} + b_m^L}}{\sum_{k=1}^{M_L} e^{(W_k^L)^T h^{L-1} + b_k^L}}\,, \tag{3.2}$$

and corresponds to the class score for the m-th class.

The loss function is then the negative log-likelihood (presented in Section 2.2.2) of the last layer

$$\mathcal{L}_{X,Y}(\theta) = -\sum_{i=1}^{N} \log P(y^{(i)}|X,\theta)\,, \tag{3.3}$$

where X, Y designs the training data and θ the parameters of the whole network

$$\theta = \bigcup_{\ell=1}^{L} \theta_\ell \quad \text{with} \quad \theta_\ell = \{W_\ell, b_\ell\}\,.$$

There are several activation functions used for the hidden layers of a fully-connected feedforward network. Historically, the ***sigmoid*** non-linearity defined in (2.7) has been widely used for two reasons. First, it has the property (see (2.8)) that its derivate in a point depends only on the value of the function in that point, which facilitates its computation. Also, it has a biological interpretation that relate it to the behavior of neurons in the human brain. The hyperbolic tangent non-linearity, denoted by ***tanh*** has a shape similar to the sigmoid and has been also widely used. A major drawback of these two non-linearity is that they ***saturate*** for large input values, which means that the gradient at these values is almost zero. This is undesirable since it prevents the network from a good learning (see the discussion about back-propagation in Section 3.1.2 for more details on this).

The rectified linear unit (**ReLU**) is defined by

$$\sigma(t) = \max\{0, t\}\,,$$

has been widely used in the recent years (see [BGC15]). It has been showed (see [ZRM+13, KSH12]) that it can improve generalization and make training of deep networks faster and simpler. In particular, it is straightforward to compute the value of ReLU units and they don't saturate as the sigmoid units.

3.1.2 Training and practical considerations

Backpropagation

Backpropagation is a way of computing gradients of expressions through recursive application of the chain rule. We now explain this procedure that consists in a forward and backward pass.

We first remark that since the activation on the ℓ-th layer of a network is a function of the activation on the $(\ell-1)$-th layer, using the chain rule we may write

$$\frac{\partial h^\ell}{\partial \theta}(\theta) = \frac{\partial h^\ell}{\partial h^{\ell-1}}(h^{\ell-1}(\theta)) \cdot \frac{\partial h^{\ell-1}}{\partial \theta}(\theta) \tag{3.4}$$

for $\ell = 0, \ldots, L+1$ with the assumption that h^{L+1} is the loss of the model and h^0 is the identity function. Now, suppose that we have a parameter θ_k in the k-th layer. Equality (3.4) implies that

$$\frac{\partial L_{X,Y}}{\partial \theta}(\theta_k) = \prod_{r=k}^{L+1} \frac{\partial h^r}{\partial h^{r-1}}(h^{r-1}(\theta_k)) \,. \tag{3.5}$$

The previous expression shows that we may first compute the values of the activations for the units in layers k to $L+1$ (the ***forward pass***), and then evaluate these values in the derivative in the reverse order (the ***backward pass***).

In Figure 3.4 we see a graphic representation of the ***information flow*** in this process. The forward pass computes values from inputs to output (shown in green). The backward pass then performs backpropagation which starts at the end and recursively applies the chain rule to compute the gradients (shown in dashed blue). The gradients can be thought of as flowing backwards through the circuit.

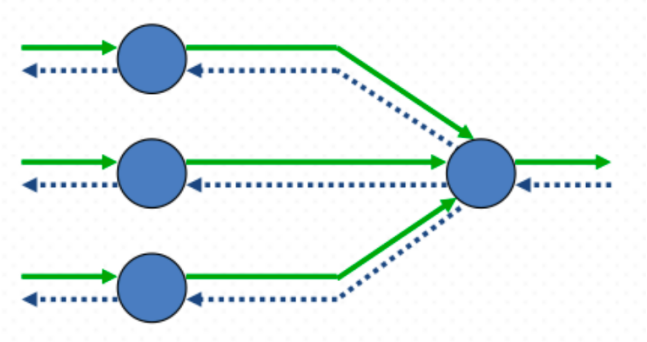


Figure 3.4: A schematic representation of information flow in the backpropagation algorithm.

Vectorized Code Optimizations

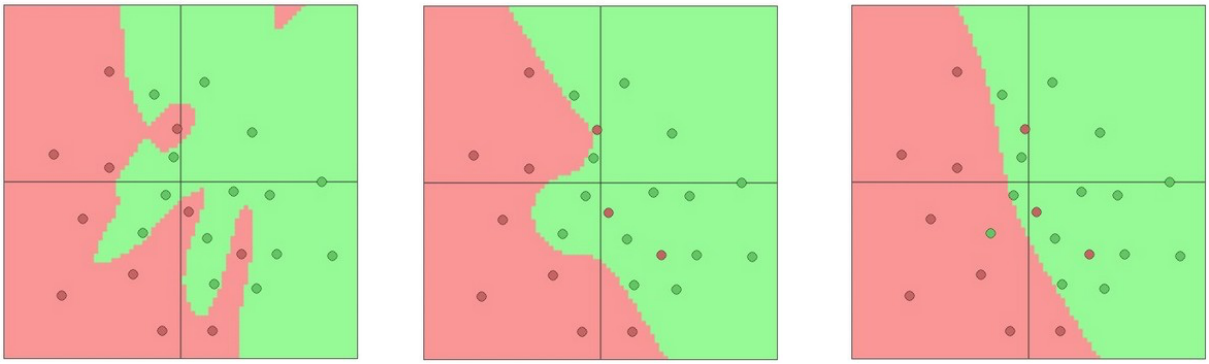
The layer-wise structure of a neural network makes it simple and efficient to evaluate the units values using matrix vector operations. Instead of having a single input column vector, we can stack samples from the training data to form inputs batches. This permits to take advantage of vectorized code optimizations so all examples will be efficiently evaluated in parallel. This is the reason why MSGD is usually preferred over SGD. Nevertheless, the size of the mini-batches should be chosen accounting to the memory constraints of the dispositive used for training.

Regularization

When adding a regularization term to the loss, we may penalize each layer independently, so

$$\mathcal{L}_{X,Y,\lambda}(\theta) = \mathcal{L}_{X,Y}(\theta) + \sum_{\ell=1}^{L} \lambda_\ell \, Q(\theta_\ell) \tag{3.6}$$

where Q is generally the ℓ^1 or ℓ^2 norm. In practice it is somehow difficult to estimate the different regularization strengths λ_ℓ, so usually they all take the same value. We notice that increasing the value of the regularization strengths should results in less complex decision regions. See Figure 3.5) for an example in simple neural network with one hidden layer.



(a) $\lambda = 0.001$ (b) $\lambda = 0.01$ (c) $\lambda = 0.1$

Figure 3.5: Decision regions learned by a neural network with one hidden layer of 20 units and different regularization strengths. Binary classification problem in two dimension. Images from [Kar].

3.1.3 Representational power and space transformation

Despite the excellent results achieved by neural networks in classical machine learning task (such as speech recognition or image classification, [HDY+12, Ben09]), theoretical results have been scarce[1]. Moreover, the complexity of a neural network makes it difficult to intuitively understand how they achieved these results. As a way to gain some knowledge about neural networks, we show here some results and example of low-dimensional simple networks (this is with a few units in each layer or with only one hidden layer).

Representational Power

An interesting question that has some theoretical answers is the representational power (as defined in Section 2.1.2) of fully-connected neural networks. This has ben addressed in [Cyb89], where the authors study neural networks with a single hidden layer and an arbitrary continuous sigmoidal function (as defined in Section 2.2.3). They show that any collection of compact, disjoint subsets of R^N can be approximated with arbitrary precision by a network with one hidden layer and a discriminatory (see here below) non-linearity.

We define as ***discriminatory*** any sigmoidal non-linearity such that the integral

$$\int_{[0,1]^N} \sigma(w^T x + b)\, dx$$

cannot be zero for all $w^T \in \mathbb{R}^N$ and $b \in \mathbb{R}$. The formal statement is then:

Theorem Let σ be any continuous discriminatory function. The finite sums of the form

$$H(x) = \sum_{j=1}^{J} \alpha_j \sigma(w_j^T x + b_j)$$

are dense in $C^0([0,1]^N)$. In other words, given any $f \in C^0([0,1]^N)$ and $\varepsilon > 0$, there is a sum $H(x)$ of the form above, for which

$$|H(x) - f(x)| < \varepsilon \qquad \forall x \in [0,1]^N .$$

[1] This seems to be changing in the last years, where formal relations between neural networks and statistical physical systems had been established, see for example [CHM+15].

Space Transformation

Looking at the form of the equalities (3.1) with (2.6), we may see that what a single unit in a layer is "simply" a non-linear classifier, similar to the logistic regression that we saw in Section 2.2.3, but with the difference that the sigmoid function may be replaced by another non-linearity. Intuitively, the representational power of neural networks comes from the composition of a large number of these layers, leading to a sequence of ***complex transformations*** of the input space. Indeed, within each layer the network transforms the input data, creating a new ***representation*** of it. See Figure 3.6 for a graphic example.

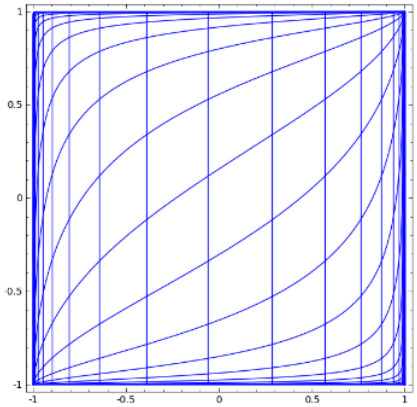


Figure 3.6: Transformation of the a square grid using a single layer with the tanh non-linearity. Image from [Ola].

Let's give a concrete example of the idea. Consider a neural network with two-dimensional inputs and one hidden layer. Let C be the square grid $[-1, 1] \times [-1, 1]$. In the fist layer, each sample point $x \in C$ is transformed by the formula $\sigma(W^T x + b)$ (see Figure 3.6). Then, since the last layer is a softmax classifier (see Section 2.2.3), it settles an hyper-plane that separates the data in two classes. We see that what this simple network does is to transform the input data into a more complex representation that can be efficiently processed with a linear classifier (see Figures 3.8 and 3.7).

Decision Regions

When increasing the depth of a network, the representation learned by the model should become more and more complex, leading to a final representation suitable for classification (see Figure 3.7). Also, we notice that as we increase the size of the layers in a network (i.e. the number of units in the layers), the representational power of the network increases since the units can "collaborate" to express more different functions (see Figure 3.9).

3.2 Convolutional Neural Networks

We now present convolutional neural networks (CNNs) which are among the best performing systems in classification/recognition tasks [Ben09]. We first explain the connectivity organization of convolutional layers and the max-pooling operation, which are the basis of the CNN structure.

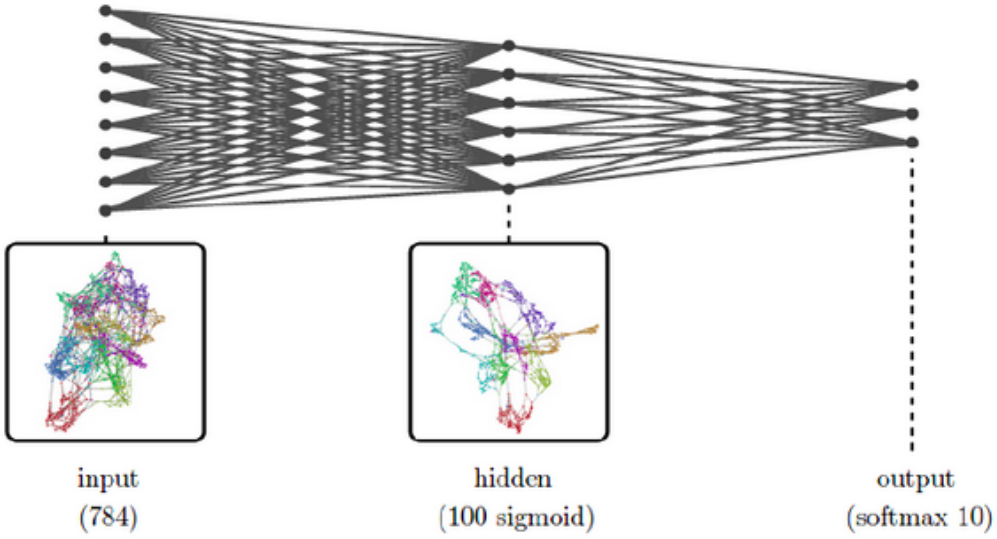


Figure 3.7: Representation learned by a neural net with one hidden layer of 100 units. The input dataset (MNIST digit images) is transformed so that the classes (each shown with a different color) can be more easily classified by the output layer. Images from [BGC15].

We then give an invariance property of CNNs that makes them very useful for recognition and classification tasks.

3.2.1 Convolutional Layers

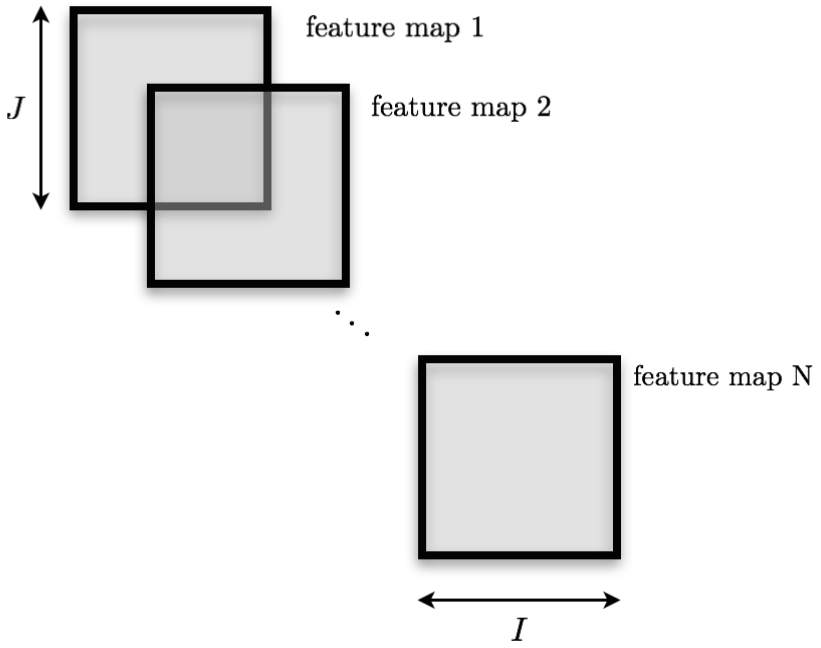


Figure 3.10: A single convolutional layer.

In a ***convolutional layer*** units are organized as stacks of two-dimensional data called ***features maps***. This is shown in Figure 3.10. A convolutional layer is thus a 3-dimensional tensor

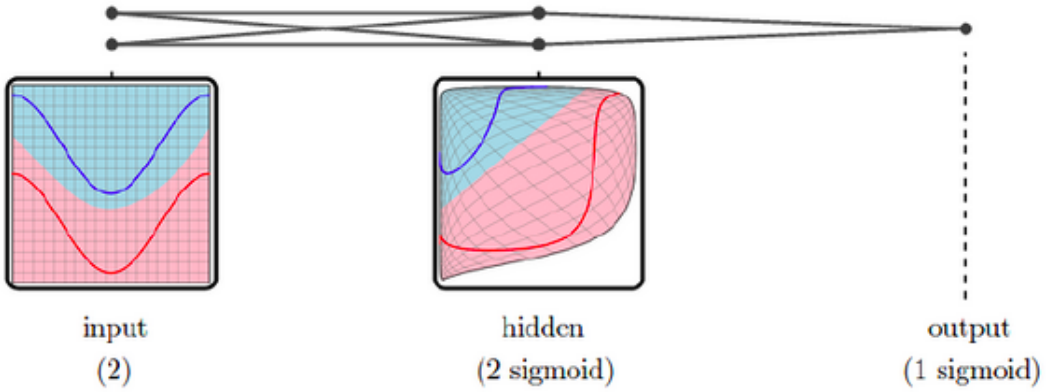
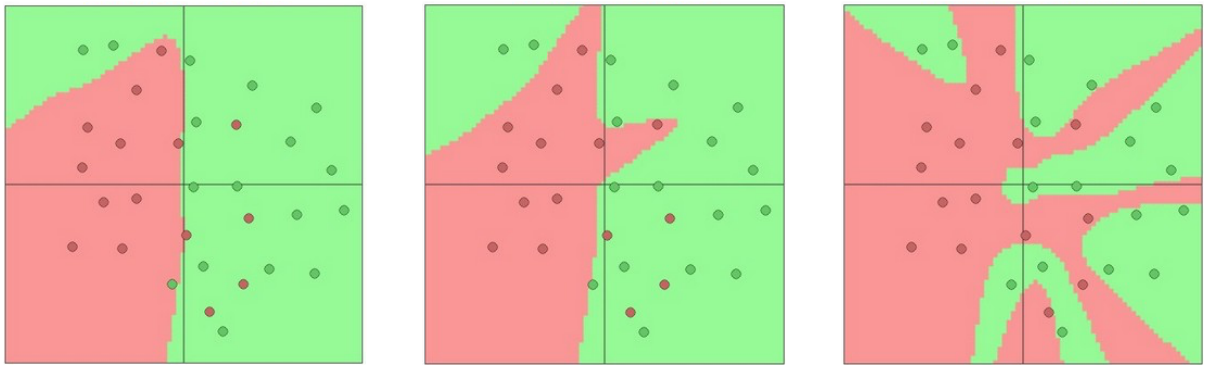


Figure 3.8: Linearly separable representation learned by a neural net with one hidden layer of two units. Images from [BGC15].



(a) Hidden layer with 3 units (b) Hidden layer with 6 units (c) Hidden layer with 20 units

Figure 3.9: Decision regions learned by neural network with one hidden layer of different size. Binary classification problem in two dimension. Images from [Kar].

$h \in \mathcal{T}_{N,I,J}(\mathbb{R})$ where N is the number of features maps while I and J are respectively the ***width*** and ***height*** of the maps. We say that (N, I, J) is the ***shape*** of the layer. A ***convolutional neural network*** is a feedforward network with several convolutional layers followed by fully-connected layers. See Figure 3.11 for a graphical example.

In Figure 3.12 we present the connectivity structure of two adjacent convolutional layers. If the ℓ-th and the $(\ell-1)$-th convolutional layers have respectively M and N features maps, the weights that connects this two layer is 4-dimensional tensor $W \in \mathcal{T}_{M,N,T,F}(\mathbb{R})$ and we have M biases $b_m \in \mathbb{R}$ (see Figure 3.13a). With each output feature map m in the layer ℓ, are associated N matrices $W^{m,n}$ of size $T \times F$ (see Figure 3.13).

Suppose that $h_\ell \in \mathcal{T}_{M,I',J'}(\mathbb{R})$ and $h_{(\ell-1)} \in \mathcal{T}_{N,I,J}(\mathbb{R})$ and let σ be the non-linearity of the ℓ-th layer. The formal expression for the activation of the unit in the position (i, j) of the m-th feature map in the ℓ-th layer is given by

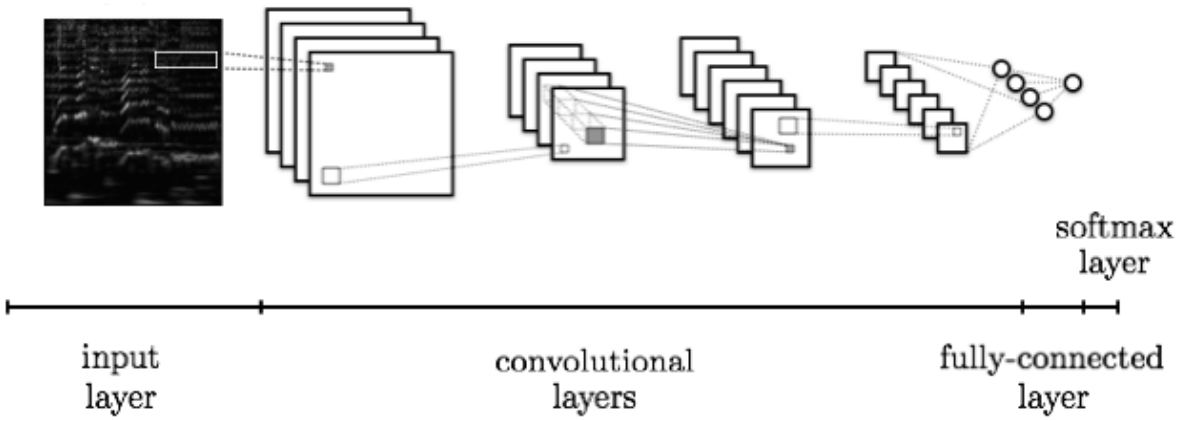


Figure 3.11: The structure of a convolutional network with five hidden layers: four convolutional followed by one fully-connected.

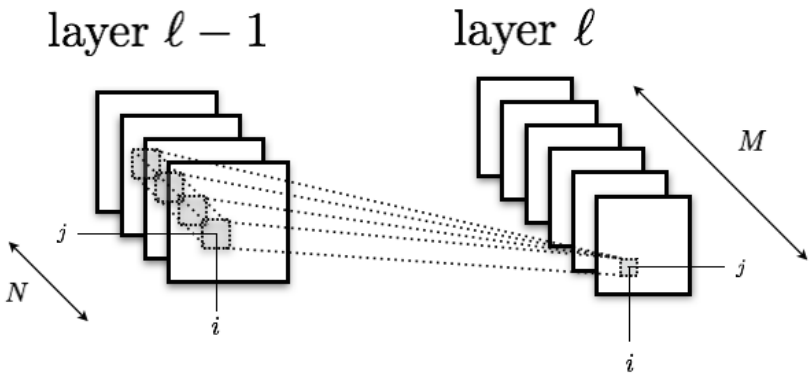


Figure 3.12: Connected regions of two adjacent convolutional layers of a CNN.

$$(h_m^\ell)_{i,j} = \sigma\left(\sum_{n=1}^{N}(W_\ell^{m,n} \star x^n)_{i,j} + b_m^\ell\right), \tag{3.7}$$

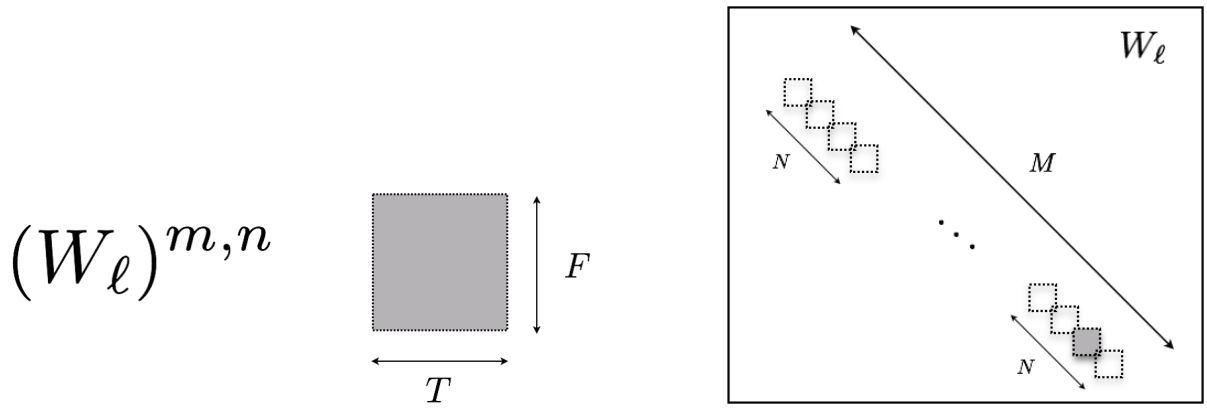
where $\star$ is a $2d$ discrete convolution (see below). Notice that we first perform $2d$ convolutions between filters and input maps with the same index n, and then we sum across the N feature maps.

We remark from (3.7) a very important property: the activation of all units in the same output map m are computed using the same weights and biases $W_\ell^{m,n}$, b_m^ℓ. We say then that the units in a output map ***share*** the weights and biases.

When considering the ***valid*** $2d$ convolution the definition for $\star$ is

$$(A \star B)_{i,j} = \sum_{r=1}^{T}\sum_{s=1}^{F} A_{r,s}B_{r+i-1,s+j-1} \tag{3.8}$$

for $A \in \mathcal{M}_{T,F}$ and $B \in \mathcal{M}_{I,J}$ with $1 \leq T \leq I-1$ and $1 \leq F \leq J-1$. We may also consider the ***full*** and ***same*** $2d$ convolutions which involves padding B with zeros in the standard way.



(a) A filter in the ℓ-th layer.

(b) The weights of the ℓ-th layer.

Figure 3.13: The structure of the weights of a convolutional layer

3.2.2 Pooling

The output feature maps of a convolutional layer are reduced in size by an operation called ***pooling***. This downsampling consists in first partitioning the maps in regions of the same size, and then extracting one value from each of the regions. When this value is the maximum, we talk about ***max-pooling***. The factor in which the map is reduced is called ***pooling size***. In Figure 3.14 we show this process.

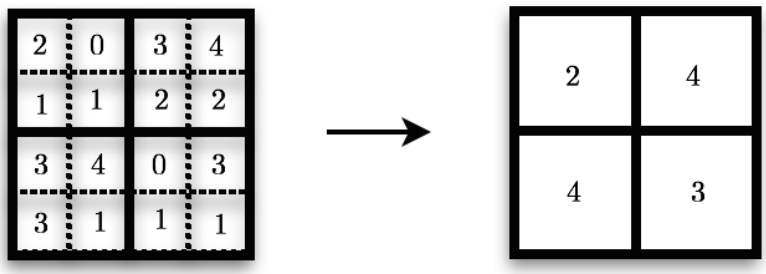


Figure 3.14: Max-pooling with of a factor 2 of a map.

3.2.3 Properties

As we saw in Section 3.1.1, in a fully-connected neural network all units from two adjacent layers are pairwise connected. On the opposite, convolutional neural networks are designed to have ***sparse interactions***, in the sense that units in an output layer are connected only to a small region of the input layer (see Fifure 3.12). This gives more efficient computations, while pooling allows units at each successive layer to take a larger part of the input into account [SB14, BGC15].

The use of shared weights within a feature map and the max-pooling operation allow the architecture of a CNN to develop an invariance to spatial translations of the input data [Ben09, HBL12]. This ***invariance property*** of a CNN can be very useful, since in many cases

the final classification should be robust to this kind of translations. This is the case, for example, when we desire to detect the presence of some pattern in the input data, but it is irrelevant ti precisely detect where this pattern is present.

CNNs have been widely used in image classification task, performing remarkably. One of the reasons usually exposed to explain this success is the following: the connectivity structure of a CNN permits the units in the deeper layers to indirectly interact with large portion of the input data. This permits the network to efficiently describe interactions between different elements in the input data. This is done by constructing such interactions from simple building blocks that each layer describe with sparse interactions [BGC15].

In Figure 3.15 we see an example of the filters learned by a CNN used for an image classification task in [KSH12]. We see that the network learn frequency and orientation selective filters (that resemble strongly to Gabor filters), as well as various colored spots. As we mention before, this kind of behavior seems to be a general property of CNNs, where high level representations can be obtained by hierarchically breaking larger systems into simpler parts. In this way, filters on the first hidden layer of a CNN appear to act as pattern detectors, capturing low level attributes of the input data.

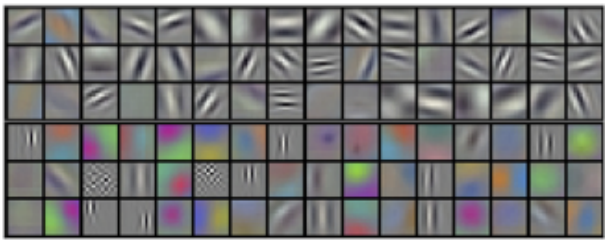


Figure 3.15: Example of the filters learned by a CNN used for an image classification task in [KSH12].

Chapter 4

Convolutional Neural Networks for Audio Classification Tasks

4.1 CNNs and audio data invariance

This internship aims to develop an automatic audio content classification system using deep neural networks. We choose to use CNNs to take advantage of both the structure of audio signals and the invariances properties that these networks may develop. We explain here the details of this approach.

Suitable representations of audio –such as the STFT or the CQT– often reveal the intrinsic attributes of audio signals that made them perceptually recognizable. For example, singing is performed with a diversity of vocal timbres, intensities, languages, etc., but it has a characteristic structure in the audio transforms (think about vibrato for instance). This structure may appear in the representations after some kind of "transformations". For example, the same sung sound can be executed at different moments (***temporal translation***, see Figure 4.2), with different intensities (***amplitude scaling***) or durations (***temporal scaling***).

As we explained in Section 3.2.3, well trained convolutional neural networks appear to work as a pattern detector in its lower layers, organizing then the information extracted in higher layers. Moreover, CNNs seem to be capable developing certain invariances under spatial translation of these patterns.

In order to take advantage of the transformation of audio mentioned here before, we choose to use the STFT and the CQT as input features. Indeed, both of these transforms are invariant under temporal translation, in the sense that two similar audio events present at different moments will appear translated horizontally (see Figure 4.1). Also, these two representations are invariant under time scaling, in the sense that the transform of an audio event performed for a long time will be locally similar. Moreover, a very important characteristic of the CQT-transform (see [Bro91, Pra10]) is that it is almost invariant under ***frequency shift***, in the sense that the transform of a sound transposed will appear translated vertically with a very close shape, see Figure 4.2.

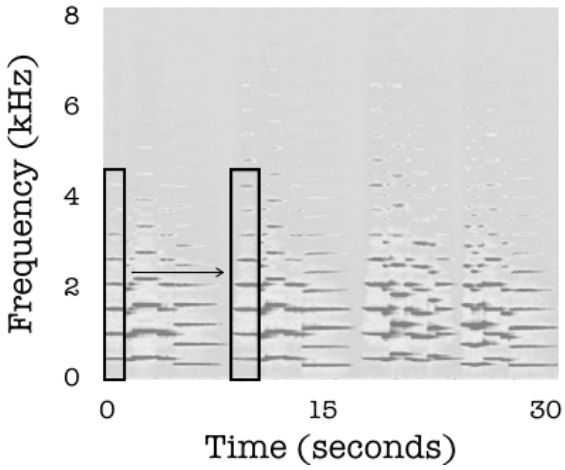


Figure 4.1: Spectrogram of the introduction of *Godfather Waltz*. We see the correspondence between the temporal translation of a note and its spatial translation in the spectrogram. Image from [Hen11].

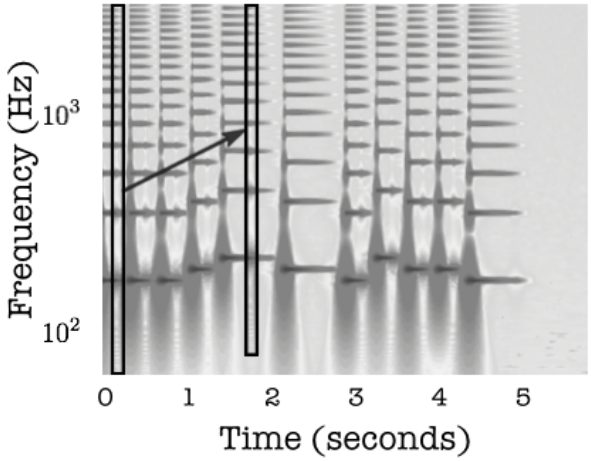


Figure 4.2: CQT-gram of "Au clair de la lune". We see the correspondence between a transposed note and its vertical translation in the CQT-gram. Image from [Hen11].

4.2 Experiments

In this section we present the experiments conducted during this internship related to male vs female voice and speech vs music classification task.

We have conceived a number of experiments to deal with the practical difficulties that arise when implementing classification systems. For example, since training a neural networks takes considerable time, we need to guess good hyper-parameters (such as the architecture of the network) before applying to real-scale situations. Also, in order to obtain a scalable model we need to use compact features and test their performance on small datasets. Furthermore, since it is very important to be certain that our systems does not overfit and generalizes correctly, we need to understand what the system is learning, for example by looking at the filters. In this aspect, we emphasize that it is not our goal to obtain the best possible classification score, so we have focused our attention primarily in obtaining a reliable and controllable classification system.

We first present the technical implementation of the CNNs as well of the common methodol-

ogy used for training and monitoring the networks. Then, we detail the experiments conducted (dataset used, input features and network structure) and the results obtained. Finally, we present some analysis on the behavior of the networks.

4.2.1 Training methodology

The input feature maps were computed using Deezer audio processing library, which is based upon the Yaafe audio features extraction toolbox from Télécom ParisTech [MEF+10].

The input data was divided in training, validation and test dataset. The loss function of our models is the negative log-likelihood of the train dataset as defined in (3.3). We minimize it using the MSGD method presented in Section 2.3.1. The performance of the network is measured during the training using the zero-one loss function over the validation dataset as defined in (2.14). To evaluate the final performance of our system, we compute the recall, precision and F-measure over the test dataset as explained in (2.15).

We have implemented several methods to ***monitor*** the behavior and performance of the model during the training phase. This is extremely important given the considerable length of the training, which depending on the size of the network and the input data, can range between some hours and some days. We typically monitor –and store for later analysis– the values of the train cost and error at the current batch, the evaluation error and the best evaluation error. It is important to verify that these decrease in mean over an epoch, which should –in principle– indicate that the parameters of the model are well settled. In Figure 4.3 we show an example of a typical training process, where we see the effect of minimizing the loss over the mini-batches and the mean decay of the errors and of the loss.

At the beginning of the training we store the initial state of the network (i.e. the value of the weights and biases) and we update these at each best evaluation. At the end of the training we store both the last and the best state of the network for further analysis and classification.

We use Theano Python library [BBB+10] to build and train our CNNs. Benefits of using Theano include symbolic computations to implement gradient-based optimization techniques. For example, the loss is implemented as a function that takes symbolic variables and its gradient computed formally. The iterative training is then carried out by assigning to these symbolic variables the values of the current batch. Also, high-cost computations (such as products of big size matrices) are accelerated by making use of the GPU and CUDA parallel computing platform. We train our network using a GeForce GTX Titan card.

4.2.2 Male vs Female Voice Classification

Experiment 1. ***Male vs Female Voice Classification.***

The first problem addressed in this internship is the classification of speaker's gender in speech recordings. The total duration is of approximatively 150 minutes. Since we used a quite small dataset, this is somehow a "toy" exercise that allowed us to validate our computational system to scale later to larger databases.

The training data is composed of portions of different speech datasets (cmu arctic [KB04], gtzan speech [Tza99], timit [FDGM86], usc timit [NTR+14] and mir-1k [HJ10]), with equally

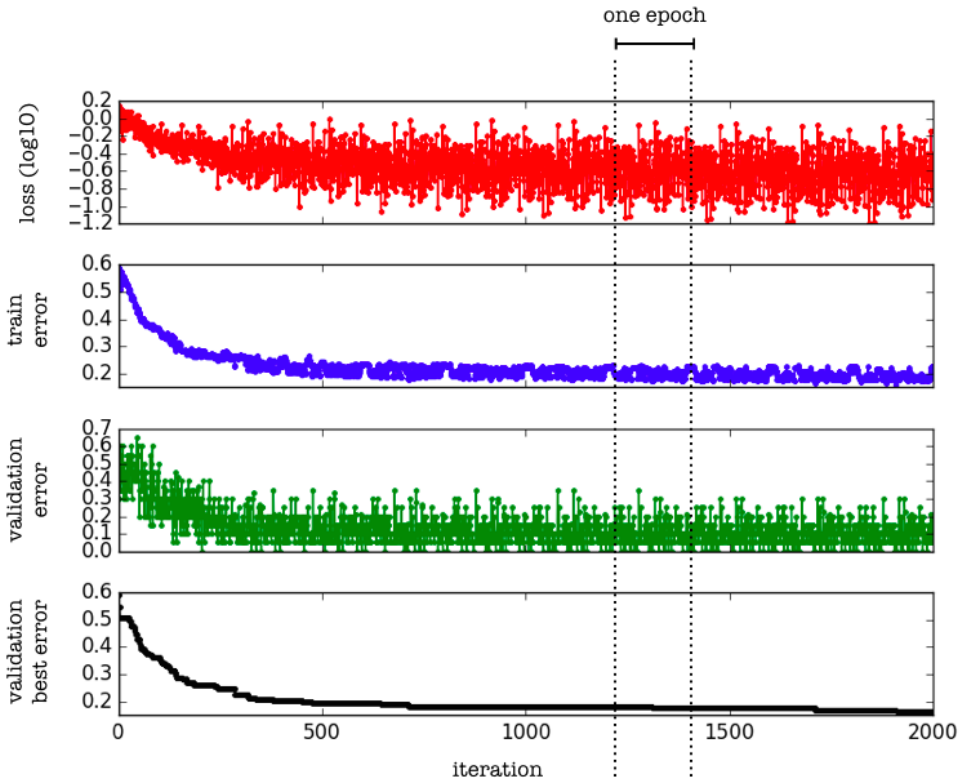


Figure 4.3: Typical evolution of the values monitored during the training phase of a CNN. Each iteration correspond to one step in the MSGD process.

distributed male and female speakers. The input features maps consist of magnitude spectrograms obtained with a hop size of 10 ms and window sizes (resolutions) of 20, 40 and 80 ms. We choose these values to capture the information present in at least one phoneme [KHH+11]. To consider the information present in a larger context, we extract temporal intervals of 150 ms centered at the discrete time to classify. We used magnitude spectrograms compressed using (4.1).

The architecture of the CNN is depicted in Figure 4.4. The input layer is a tensor of shape $(3, 15, 369)$ which dimensions respectively correspond with spectrogram resolutions, temporal frames and frequency bins. After the input layer there are two pairs of convolutional and max-pooling layers. These are followed by a fully-connected layer and the logistic regression layer. Since our dataset is small, we set a low number of filters in the convolutional layers trying to prevent overfitting.

The filters in the first convolutional layer are aimed at recognizing small patterns present in the spectrograms (for example portions of phonemes). We expect the filters in the second convolutional layer to capture more high level attributes of speaking voice. Also, we use rectangular filters (narrow in time and wide in frequency) to capture changes in the spectral content of speaking voice. The max-pooling layers perform subsampling only in the temporal dimension, since – as we explained in Section 4.1– we desire an invariance of the CNN under temporal translation of audio events. The last layer has 2 output units and computes the male/female prediction using soft-max probabilities.

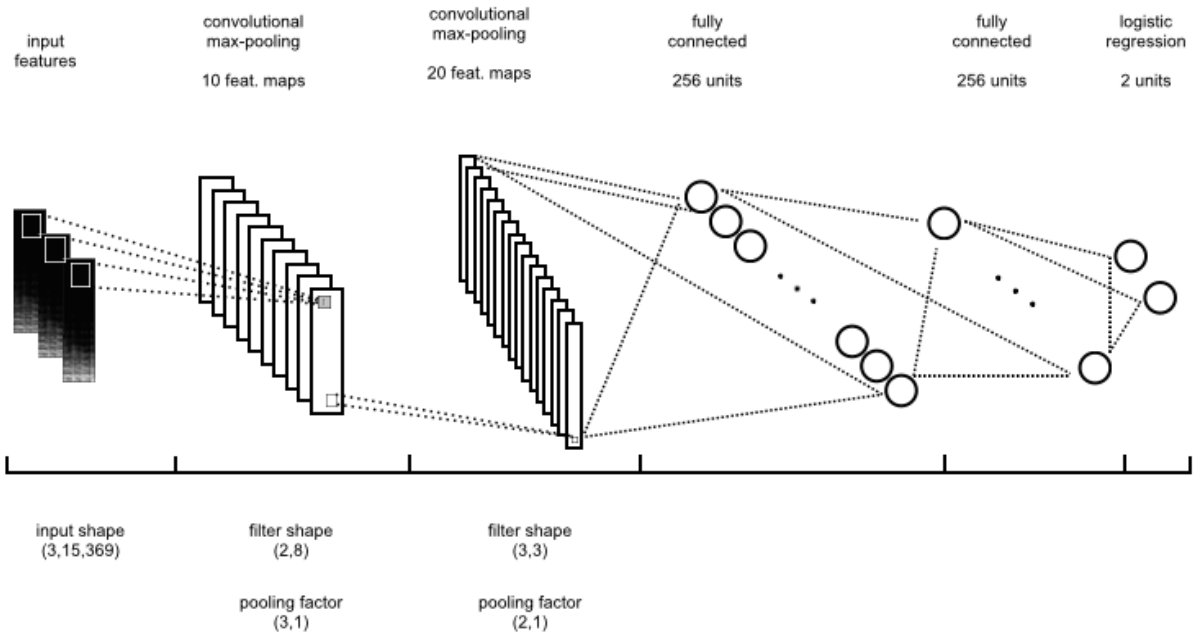


Figure 4.4: CNN architecture for experiment 1: male vs female voice classification task.

We split the dataset in 70% for training, 20% for validation and 10% for testing. When training the network with mini-batches of 100 samples, after 220 epochs and using a fixed learning rate of 0.01, we obtain an error result of 1%. The whole training took 1.46 minutes.

On a first glance our classification result is acceptable, but we have trained the network with a very small database. Probably our system overfits, perhaps because it learns the speaker's identity rather than their gender. Nevertheless, this experiment allowed to us to validate the computational architecture of the CNN and the training process.

4.2.3 Speech vs Music: Classifiers, Input Features and Architectures

The second and main problem addressed in this internship is the classification of speech vs music in Deezer database. We first present some small experiments using CNNs with different input features and architectures. Then, we present our real-scale experiments using the CQT.

The training data is composed of two classes of audio content from Deezer database. The first class, that we call ***speech***, comes from different speaking sources such as audiobooks, interviews, political speeches or pieces of theater. The second class, that we call ***music***, comes from songs of different musical styles. Speech recordings have often a sampling rate of 16.000 Hz, so we resampled the whole training data to this frequency. The two classes are equally represented, with a total of **41.960 different annotated audio sources** available for the datasets. From each audio source we extract a **randomly selected 5 seconds sample** from which we computed the input audio features. This represents in total 58.3 **hours of audio**. Since the annotations were made with semi-automatic methods, it is possible that some samples contain silences or other artifacts. Nevertheless, we have verified the accuracy of the validation and test datasets by listened to all the samples.

We used two datasets (see Table 4.1), a large one containing all the available data and a smaller one, to rapidly test some properties of the CNNs.

Except explicit mention, the features used the experiments of this section are computed as in Table 4.2. We show in Figure 4.5 two typical examples of CQT-grams for speech and music samples. In order to reduce the dynamics of some features, we compressed it using

$$f(s) = \log_2(1 + Cs)\,, \qquad \text{with} \qquad C = 10.000\,. \tag{4.1}$$

	Train	**Validation**	**Test**
Small dataset	8.000	472	472
Large dataset	40.000	980	980

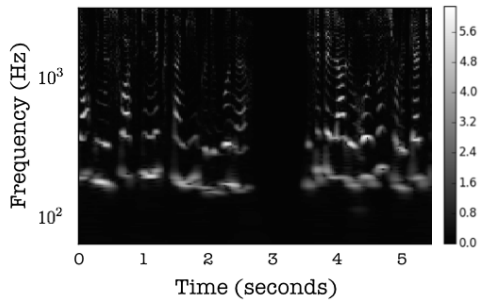
Table 4.1: Datasets for the speech vs music classification task.

CQT	Single resolution CQT-gram, computed with a window of 64 ms, 6 octaves, 24 bins per octave and a minimum frequency of 80 Hz. Compressed using (4.1).
STFT	Single resolution magnitude spectrogram, computed with a window of 20 ms and an overlap of 10 ms. Compressed using (4.1).
MFFC (M)	40 mel filters MFFC (minimum and maximum frequency respectively of 130 and 6854 Hz), computed with a window of 64 ms and without overlap. We keep 12 cepstral coefficient without the first.
CHROMA (C)	Computed with a window of 64 ms, 6 octaves. Contains 36 bins per octave and a minimum frequency of 73.42.
(C)+(M)	The aggregation of the two features here before.
(C)+(M)+Δ(M)	The same as before plus the time derivative of the MFFC.

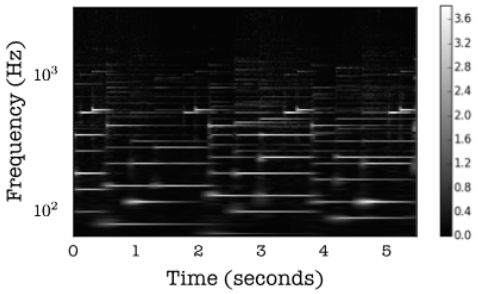
Table 4.2: Input features for the speech vs music classification tasks.

Experiment 2. ***Different resolutions magnitude spectrograms.***

We performed an experiment to evaluate the influence of using different resolutions magnitude spectrograms. The three different magnitude spectrograms (see Figure 4.6) were obtained using window sizes of 20, 40 and 80 ms, with a hop size of 10 ms and they were compressed using (4.1). We use the small dataset from Table 4.1. In Table 4.3 we see the memory resources associated with these input feature configuration and the results of the training (90 epochs in the three cases). We see that using a finer resolutions increase considerable the memory needed and the training time while softly improving the classification performance.



(a) Speech sample "Lettre d'Alfred de Musset à George Sand: Pauvre George", deezer.com/track/67664893.



(b) Music sample "Desire, Musique Douce", deezer.com/track/57136771.

Figure 4.5: Typical CQT-grams of music and speech samples.

Resolution	feat. shape	feat. mem. [GB]	training [h]	Recall	Precision	F-measure
20 ms	(1,431,700)	4.83	7.32	91.53	90.76	**91.14**
40 ms	(1,216,700)	2.42	3.60	88.98	86.07	87.50
80 ms	(1,108,700)	1.21	1.80	86.44	91.07	88.70

Table 4.3: Results of experiment 2. Same CNN structure with different resolutions (window length) magnitude spectrograms.

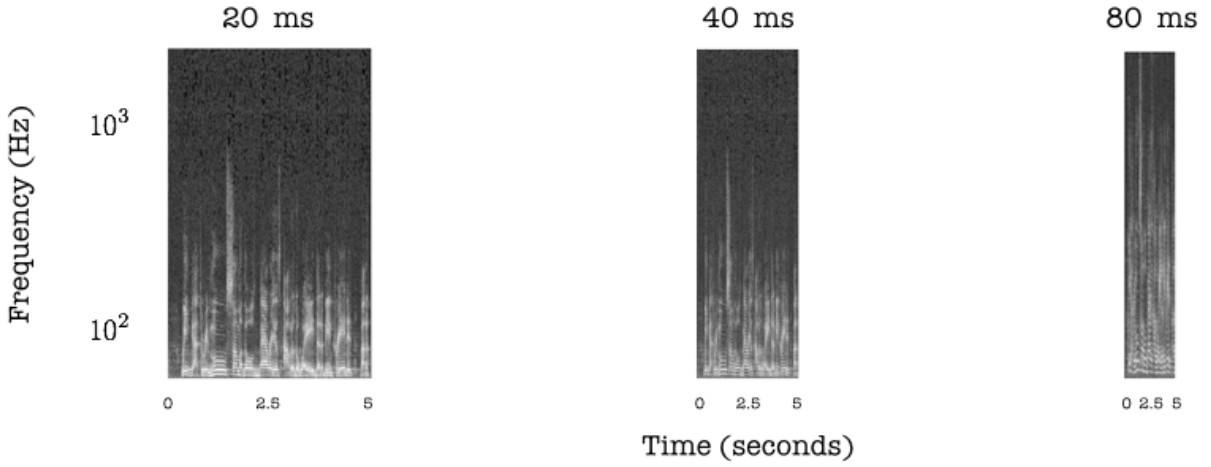


Figure 4.6: Three magnitude spectrograms of the same audio sample with different window sizes (resolutions). "Republican Independence Debate # 3: Energy Independence" speech sample, seconds 33-38, deezer.com/track/13938423.

Experiment 3. ***CNN versus other standard classifiers***

We performed an experiment to compare the CNN with two other standard classifiers: ***Random Forests (FM)*** [Bre01] and ***Support Vector Machines (SVM)*** [SS04]. We use the same input features for the three classifiers: MFFC + Δ-MFCC + Chroma computed over the small dataset from Table 4.1. For the CNN we stack the three audio descriptors in a 3d tensor, while

	Input Shape	Training [s]	Recall	Precision	F-measure
SVM	(1, 3888)	40	82.17	89.83	85.83
RF	(1, 3888)	16	76.27	89.11	82.19
CNN	(3, 108, 12)	**504**	93.10	90.00	**91.53**

Table 4.4: Results of experiment 3. Comparison between SVM, RF and CNN classifiers trained for 90 epochs with MFFC + Δ-MFCC + Chroma as input features.

for the SVM and the RF we align all the data in one vector (see first column of Table 4.4). For the CNN we use the same network architecture as in experiment 5, see Figure 4.7. We implement the RF and the SVM classifiers using Python's machine learning library Scikit-Learn [PVG+11]. We use the default implementation of Scikit-Learn for the RF and the SVM classifier, this is, 10 trees for the RF and a non-linear kernel for the SVM. The three classifiers were trained during 90 epochs. As we see in Table 4.4, the CNN classifier clearly outperforms the two others, but it takes longer to train. We remark that we didn't try to optimize the parameters of the SVM and RF classifiers, but just to have a point of reference against which compare the CNN classifier.

Experiment 4. ***Different number of layers.***

We performed two experiments to evaluate the influence of the number of convolutional layers in a CNN. In the first of them, the input features are magnitude spectrograms and in the second CQT-grams, computed over the small dataset from Table 4.1. In both experiments the CNN architecture has either one, two or three convolutional layers, followed by two fully-connected layers and the logistic regression at the end. In every case the CNN was trained during 90 epochs. In Table 4.5 we see the results of the training. In both experiments 1 convolutional layer gives a results a little better that 2 convolutional layers, but 3 convolutional layers consistently improves the classification results. Notice also that evidently adding more layer results in longer training time.

We also performed an experiment to evaluate the importance of the number of fully-connected layers in the CNN architecture. We use CQT-grams input features computed over the small dataset from Table 4.1. The architecture contains three convolutional layers, one or two fully-connected layers and the logistic regression at the end. In Table 4.5 we present the results of the trainings, which show that two fully-connected layers improves the classification results.

Experiment 5. ***Different input features.***

We performed an experiment to evaluate the influence of choosing different input features with the same CNN architecture. The architecture of the network is presented in Figure 4.7 and the details of the features used in Table 4.2. The CNN was trained during 90 epochs. As we may appreciate looking at the results in Table 4.6, both the STFT and the CQT clearly outperform the audio descriptors MFFC and CHROMA, but the aggregation of CHROMA, MFFC and Δ-MFFC performs better than the STFT and the CQT. This seems coherent with the fact that there is more information in the STFT and in the CQT that in the others audio descriptors. However, the inference time for the CQT is shorter than for the CHROMA, MFFC and Δ-MFFC since the computation of the CQT is shorter than the other features. We also notice that the performance of the network using STFT and the CQT are quite similar, but that the CQT requires less memory and training time.

STFT	Training [h]	Recall	Precision	F-measure
1 convolutional layer	3.23	93.22	83.33	88.00
2 convolutional layers	5.12	90.68	82.95	86.64
3 convolutional layers	7.32	93.10	90.00	**91.53**

CQT	Training [h]	Recall	Precision	F-measure
1 convolutional layer	1.12	94.92	91.06	92.95
2 convolutional layers	1.89	93.22	92.44	92.83
3 convolutional layers	2.03	94.92	93.33	**94.12**

CQT	Training [h]	Recall	Precision	F-measure
1 fully-connected layer	0.79	90.68	90.68	90.68
2 fully-connected layers	2.03	94.92	93.33	**94.12**

Table 4.5: Results of experiment 4. Comparison between several convolutional layers with spectrograms or CQT-grams as input features.

	Input Shape	Input Mem. [GB]	Training [h]	Recall	Precision	F-measure
CQT	(1, 216, 144)	0.50	0.84	90.68	89.92	**90.30**
STFT	(1, 431, 700)	4.83	7.32	91.53	90.76	**91.14**
MFFC (M)	(1, 108, 12)	0.02	0.13	85.59	83.47	84.52
CHROMA (C)	(1, 108, 12)	0.02	0.12	75.86	88.00	81.48
(C)+(M)	(2, 108, 12)	0.04	0.13	93.10	84.38	88.52
(C)+(M)+Δ(M)	(3, 108, 12)	0.06	0.14	93.10	90.00	**91.53**

Table 4.6: Results of experiment 5. Comparison between input features.

Experiment 6. ***Regularization***

Since the small dataset does not contains a lot of training data, it is possible that the classifiers from the previous experiments overfits. As we explained in Section 2.3.2, a way to prevent this is to add a regularization term to the loss of the model, so we performed an experiment to investigate this. We compared the effect ℓ_1-regularization when training during 100 epochs a CNN with CQT as input features. The architecture of the networks is presented in Figure 4.8. As we see in Table 4.7, the ℓ_1-regularization improved the performance, which indicate a better generalization of the classifier.

	Training [h]	Recall	Precision	F-measure
no reg.	7.76	93.22	94.02	93.62
ℓ_1 reg.	7.81	96.61	96.61	**96.61**

Table 4.7: Result experiment 6. CNN with CQT-grams as input features computed over the small train dataset with and without ℓ_1 regularization.

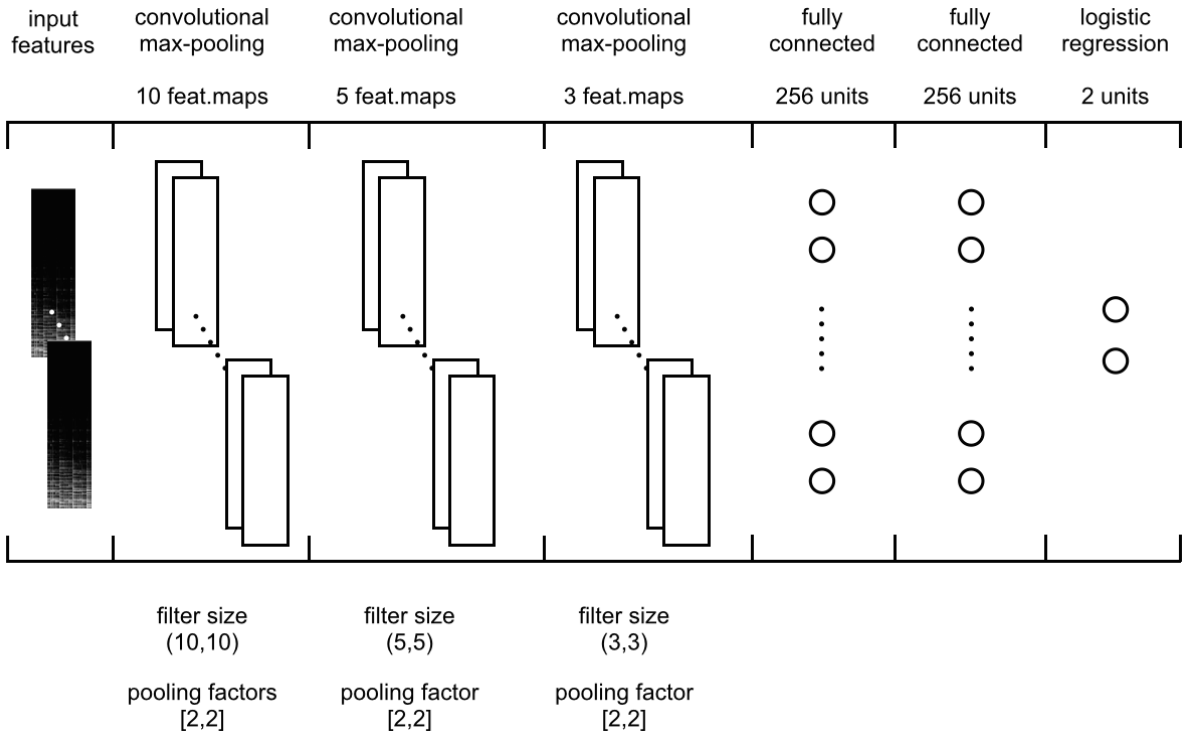


Figure 4.7: CNN architecture for experiment 5.

4.2.4 Speech vs Music: CNN and CQT

As we can see from the experiment 5, a CNN classifier with CQT-grams as input features perform well for the speech vs music task, while maintaining a relatively (with respect the STFT) low training time. Since we are interested in exploiting the invariance of the CQT (see Section 4.1), we perform some experiments using only CQT-grams as input features computed over the large dataset from Table 4.1. We also study the influence of different regularization penalties in the filters learned by the CNN.

We explain here the design of the architecture of the CNN. The filters in the first layer aim to capture frequency information, so we have settled it to correspond with 3 octaves (half of the whole CQT) and with 0.23 seconds. We then pool only in the frequency dimension. The filters in the second layer aim to capture temporal information, so we have settled it to correspond with 0.5 seconds. We then pool only in the temporal dimension. We expect the third and fourth convolutional layers to organize the information captured in the two first layers, so we put square filters and we pool alternatively in the frequency and temporal dimension. We use valid convolution in all the convolution layers. We expect the fully-connected layers to trait the information extracted before in order to perform a good classification.

In the following experiment we used the CQT-grams from Table 4.2 as input features.

Experiment 7. ***CQT - Large Dataset***

In Table 4.8 we see the results of training during 8 epochs a CNN with the architecture presented in Figure 4.8. We notice that compared to the experiment with spectrograms as input features (see Experiment 5), the CQT-grams perform better with significantly lower training

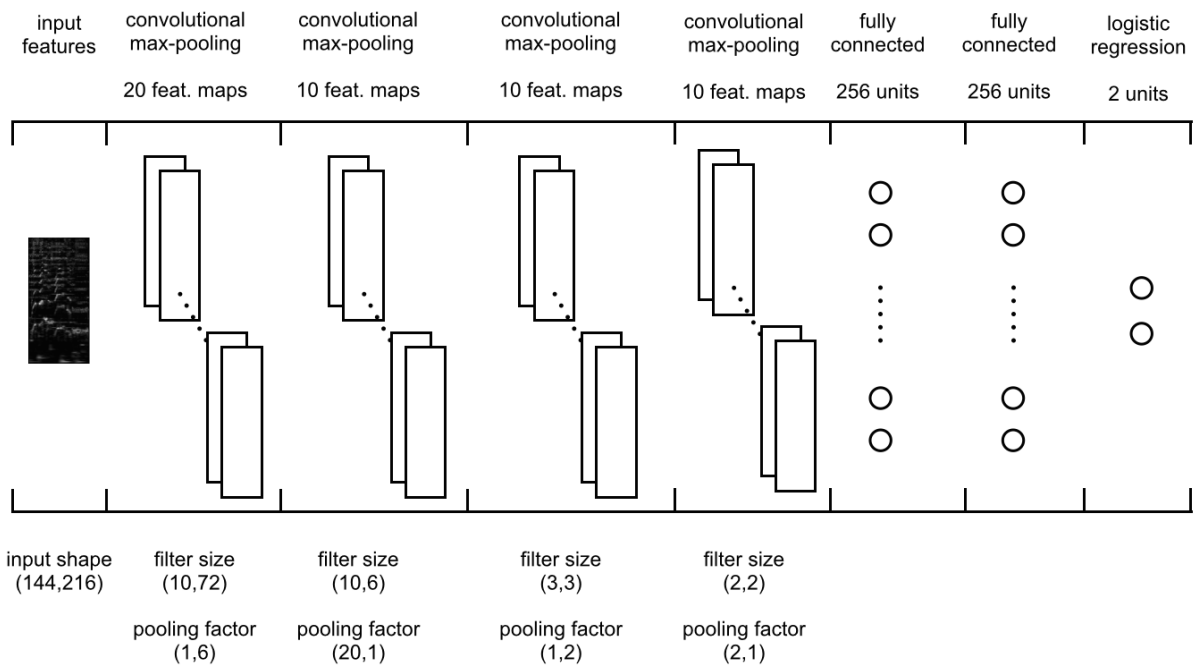


Figure 4.8: CNN architecture for experiment 7.

time and using almost 5 times less data. In Figure 4.9 we show some of the filters learned by the network in its first layer. As expected, since we didn't added any regularization to the network, these are noisy and don't have a strong recognizable structured. Nevertheless, with a little of good willing, we may perceive horizontal lines and small homogeneous areas, revealing the existence of some structure in the filters.

Input Mem. [GB]	Training [h]	Recall	Precision	F-measure
2.49	1.26	95.83	95.83	**95.83**

Table 4.8: Result experiment 7. CNN with CQT-grams as input features computed over the large dataset.

Experiment 8. ***CQT - Filters Learned***

We performed an experiment to examine the filters learned by the CNN when adding different regularization penalties. The architecture of the network is the similar to the one presented in Figure 4.7, except for the number of feature maps in each layer (the size of the filters remain the same). Looking at the results presented in Table 4.9, we see that the performance are quite similar. Nevertheless, the feature maps learned differ, see Figures 4.10-4.13. We see that in general the filters are quite structured, with arrangement of vertical and horizontal patterns. These results improve our confidence in the CNN classifiers, since it indicate that the networks do not learn random features and hence diminishing the possibility of overfitting.

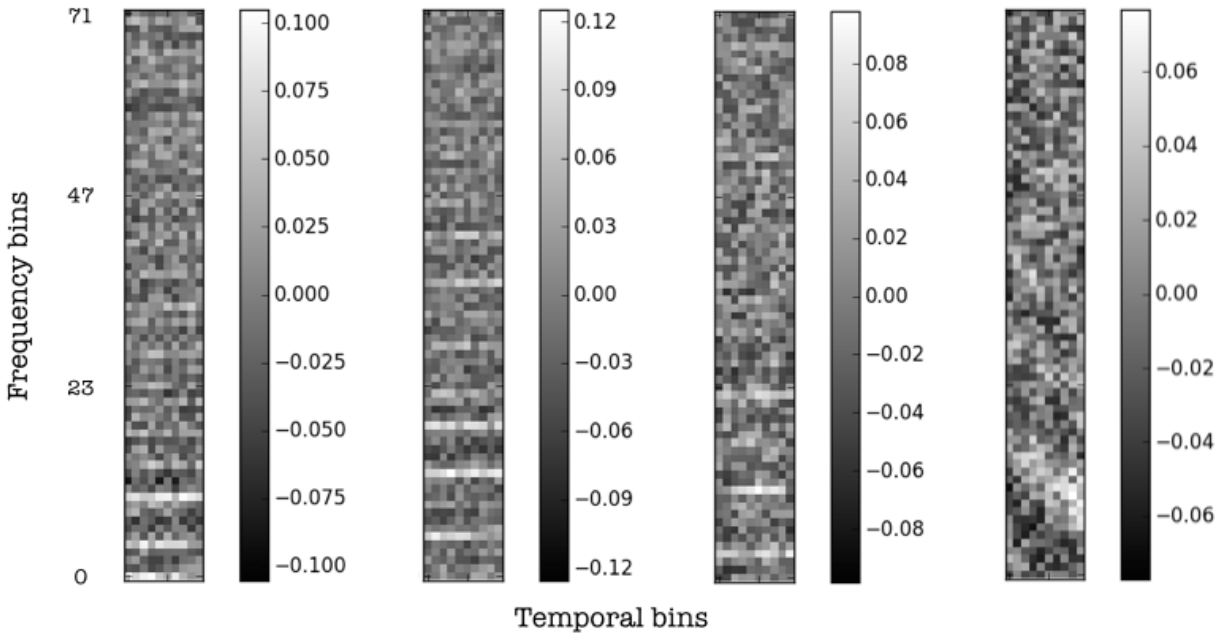


Figure 4.9: Filters learned at the first layer of the CNN from Experiment 7 without regularization.

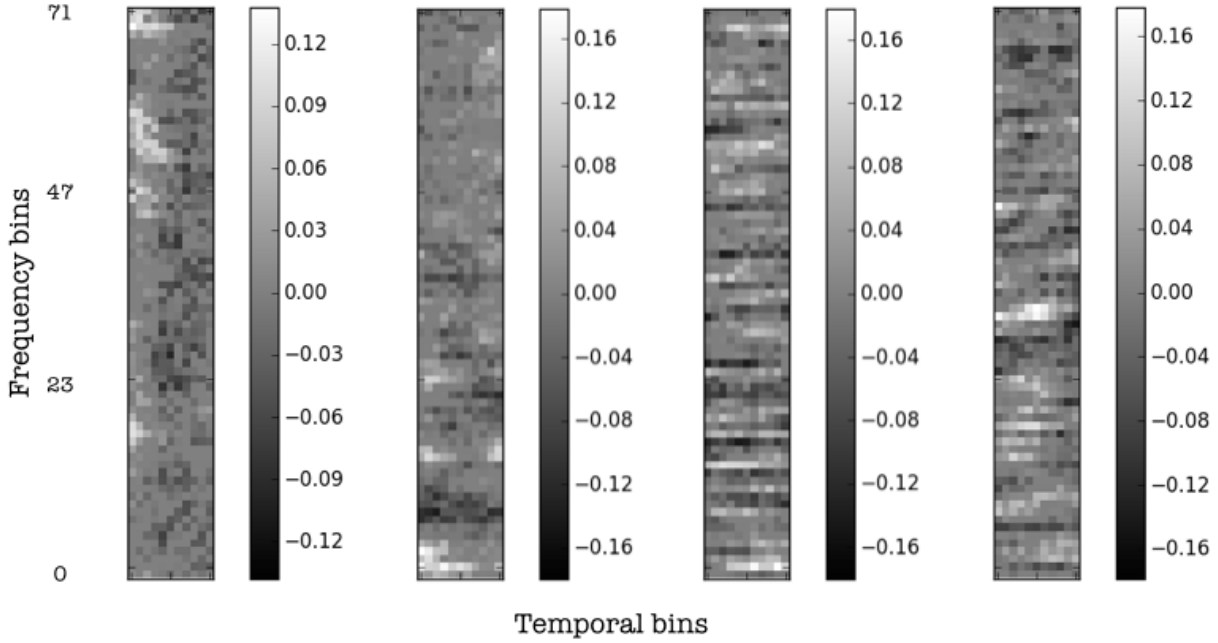


Figure 4.10: Filters learned at the first layer of the CNN from Experiment 8 with ℓ_1/ℓ_2 regularization.

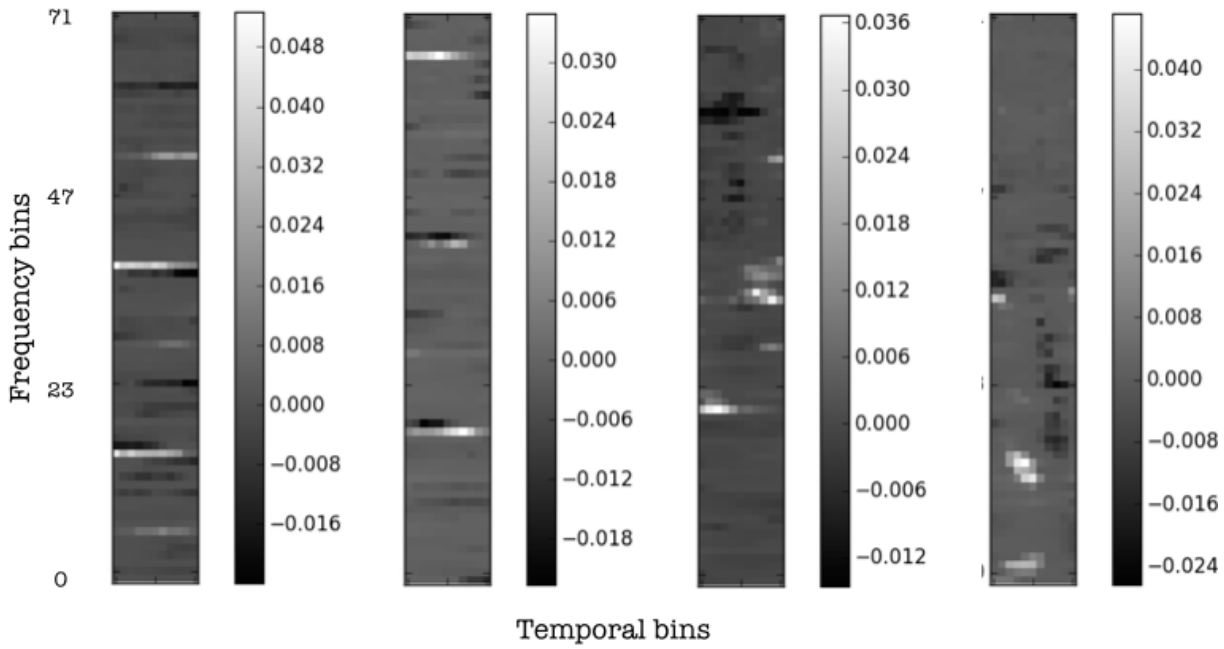


Figure 4.11: Filters learned at the first layer of the CNN from Experiment 8 with ℓ_{12} regularization.

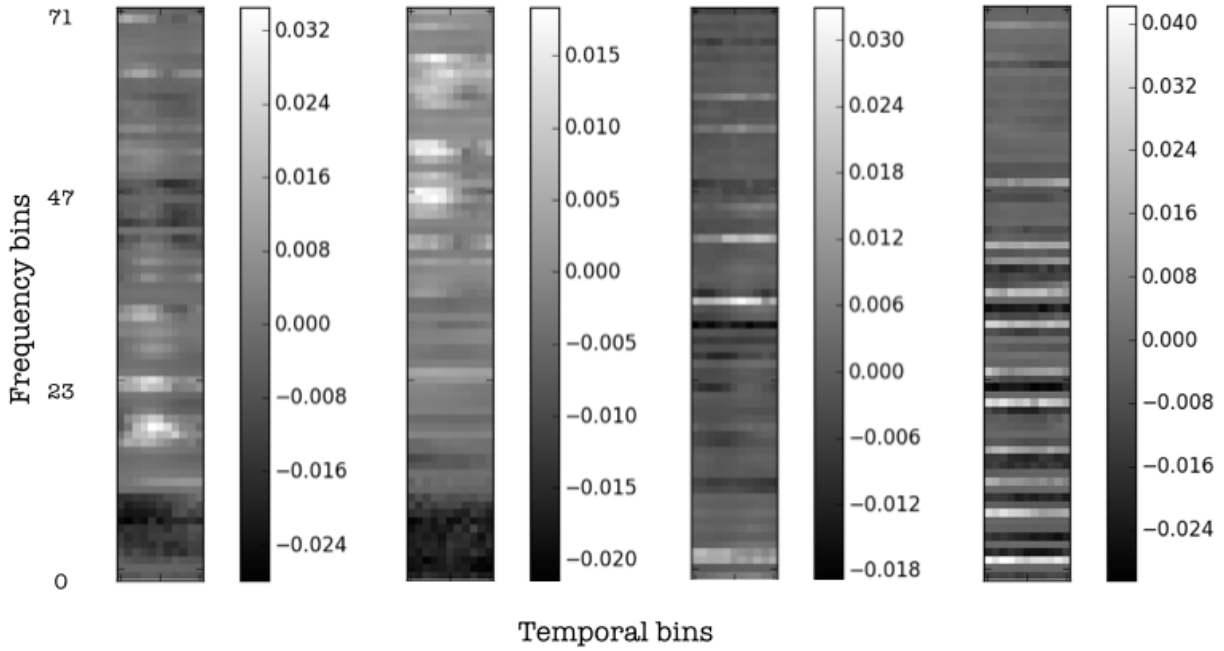


Figure 4.12: Filters learned at the first layer of the CNN from Experiment 8 with ℓ_{21} regularization.

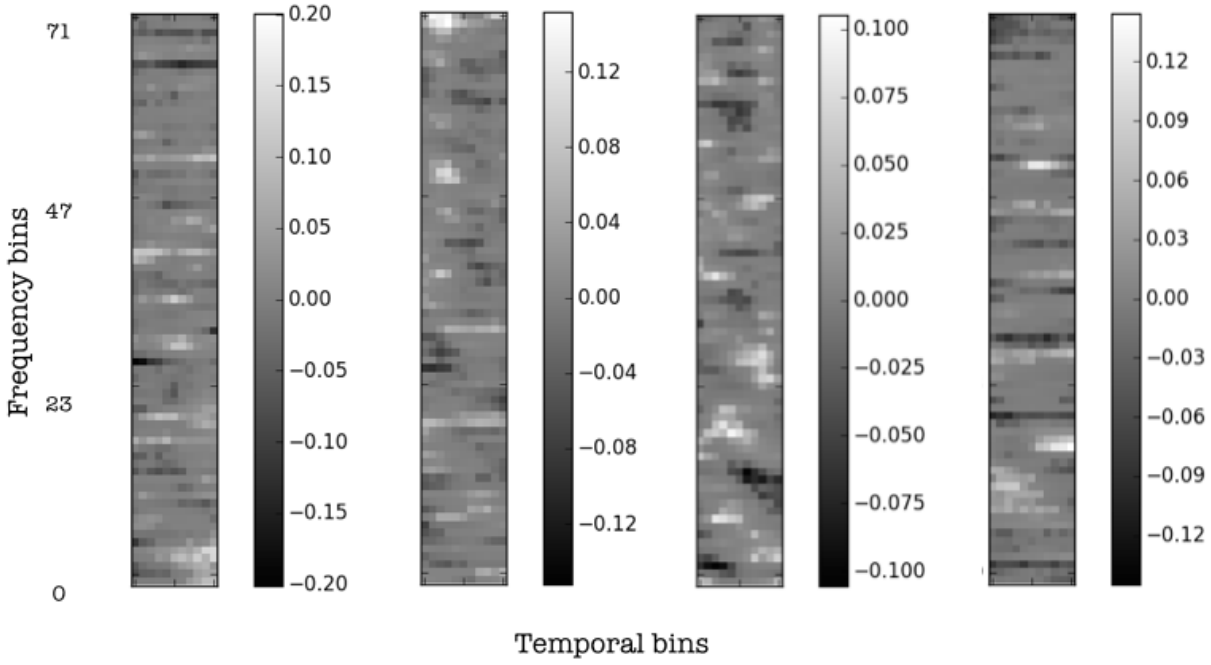


Figure 4.13: Filters learned at the first layer of the CNN from Experiment 8 with ℓ_1 regularization.

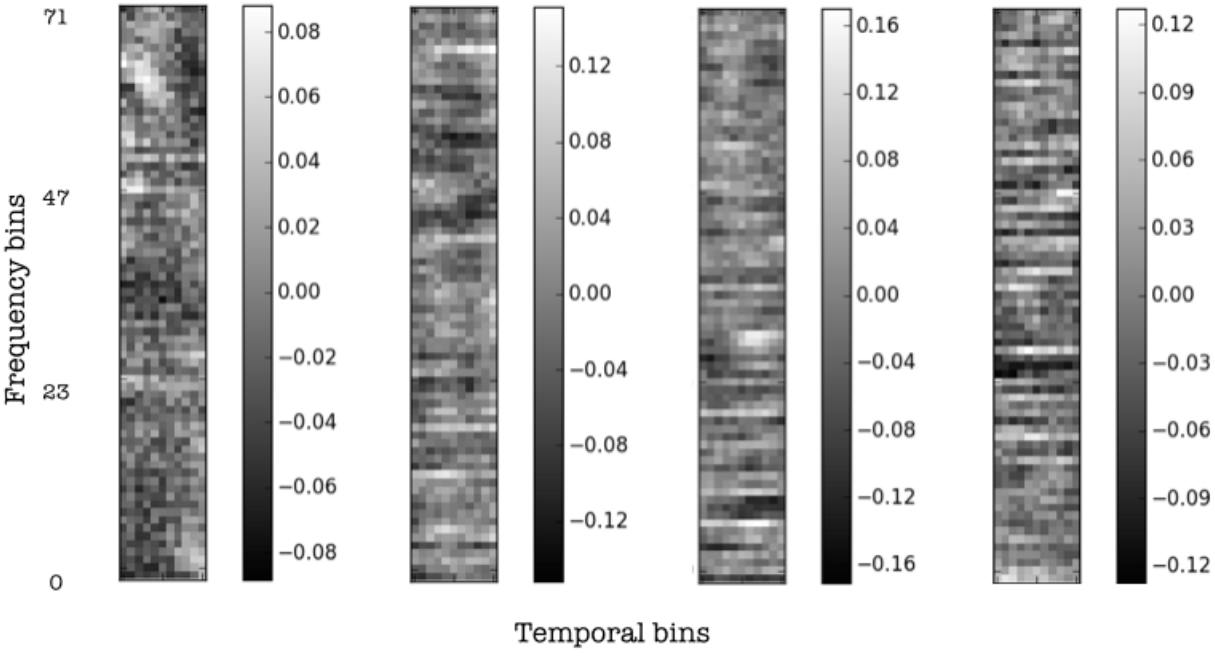


Figure 4.14: Filters learned at the first layer of the CNN from Experiment 8 with ℓ_2 regularization.

	N° feat. maps layer 0	Epochs	Training [h]	Recall	Precision	F-measure
ℓ_1	20	89	5.14	89.83	93.81	91.77
ℓ_2	20	60	5.57	92.37	93.97	93.16
ℓ_1/ℓ_2	20	20	2.18	91.53	90.00	90.76
ℓ_{21}	100	128	27.63	88.14	96.30	92.04
ℓ_{12}	100	299	64.45	92.37	96.46	94.37

Table 4.9: Result experiment 8.

4.3 Understanding the networks

The lack of theoretical results supporting the empirical success of deep neural networks make necessary the development of techniques to understanding their behavior. Indeed, in general it is not really clear why they perform so well, or how they might be improved.

One technique to analysis the functioning of CNNs consists in visualizing the filters learned by a CNN as we did in Experiment 8. We show here another technique, consisting in reconstructing the input data using the filters learned by the network.

4.3.1 Deconvolutional neural networks

The visualization technique presented here was first developed in [ZTF11] and we refer to [ZF14] for more detailed explanations. This method gives some insights into the function of intermediate feature layers of the classifier. The main idea consist in analyze the feature activity in the intermediate layers of the CNN by "mapping" these activities back to the input space. By this, we mean to sequentially invert the convolution and pooling operation at each layer. In this way it is possible to perceive what input pattern could originally cause a given activation in a feature maps.

We have applied this technique using the experiment with CQT-grams as input features. We show in Figure 4.15 the results. Once the reconstruction is achieved we may use an inversion method of the CQT explained in [GL84] to hear how much audio information is preserved in the network.

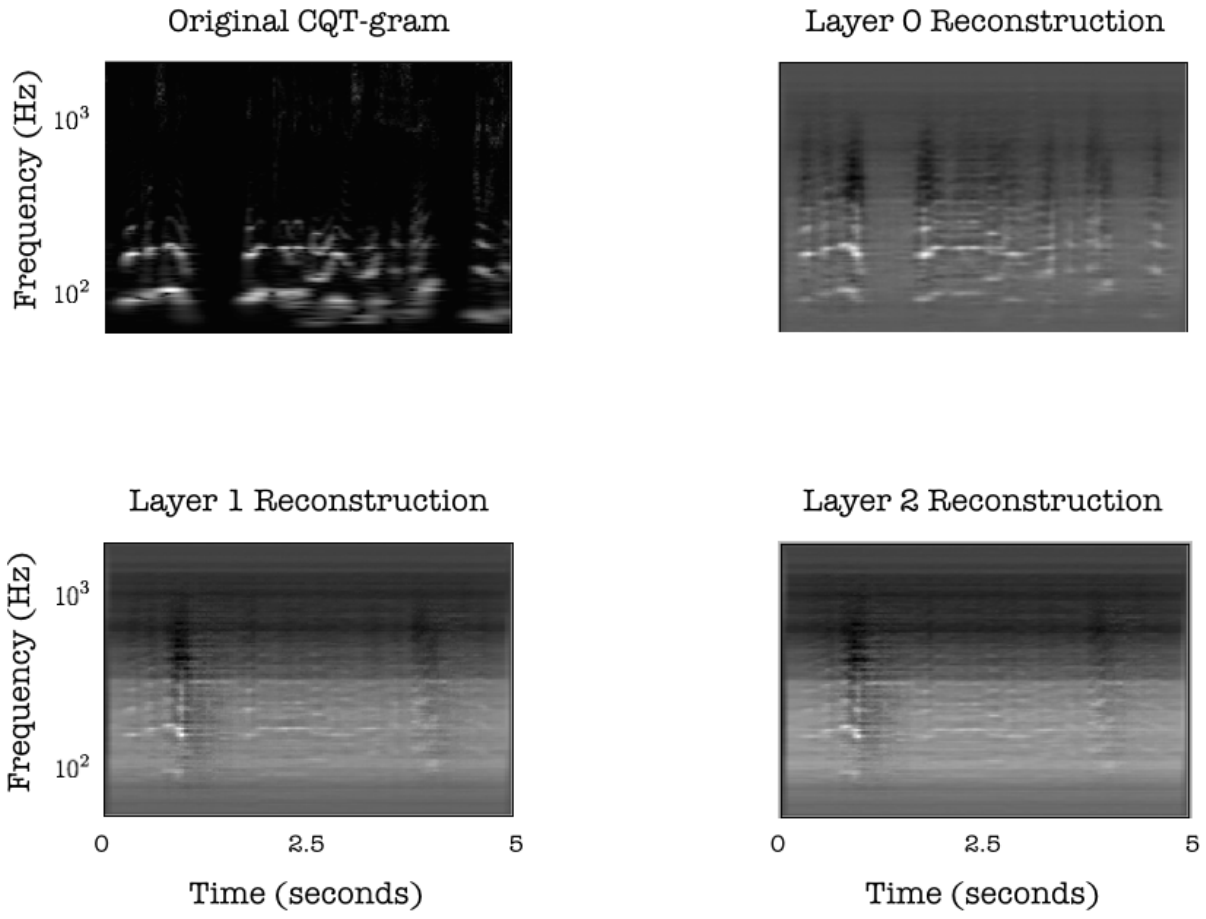


Figure 4.15: CQT-grams reconstructions from the speech sample "L'art du bonheur Chapitre 8" by Le Dalai-Lama and Howard Cutler, deezer.com/track/77525826.

Chapter 5

Applications, discussion and future work

5.1 Practical Applications

We now explain some practical applications –that we have experimentally tested– of the speech vs music classification system implemented in this internship.

5.1.1 Tag cleaning in Deezer catalog

Audio tracks in Deezer catalog are annotated using different ***tags***, such as genre, country, mood, etc. This type of metadata is extensively used in the different queries of audio content in Deezer, for example, in the context of automatic recommendation, playlists generation, etc. The reliability of these tags is thus crucial for these task. Unfortunately, the tag's sources are not always trustworthy, which derives in badly annotated audio content. In order to reverse these situations, we may use our speech vs music classification system to detect audio content that is annotate as speech, but that we classify as music, or vice versa, improving in this way the quality of the tags in Deezer catalogue. For example, using our classifier we did a **tag correction for 3.16% of the 10.000 most listened audiobooks** in Deezer, that were actually music content.

5.1.2 Song detection in audio content

Another interesting application is the **recognition of music tracks at the interior of audio content** that is majority speech. For example, we may want to detect the presence of music inside of a podcast or radio show. Also, combining our speech vs music classifier with other MIR tools such as the fingerprint (see [JYL+09]), we may first detect the musical content and then match it in Deezer catalogue. In Figure 5.1 we see an example of this idea. We have first sliced the audio of the first 50 minutes of the film "Pulp Fiction" in samples of 5 seconds. We classify then the samples as speech or music, assigning a confidence value to the classification. This value is actually the soft-max probability assigned by the last layer of the CNN. Finally, we match the sections annotated as music with tracks in Deezer catalogue using the fingerprint algorithm.

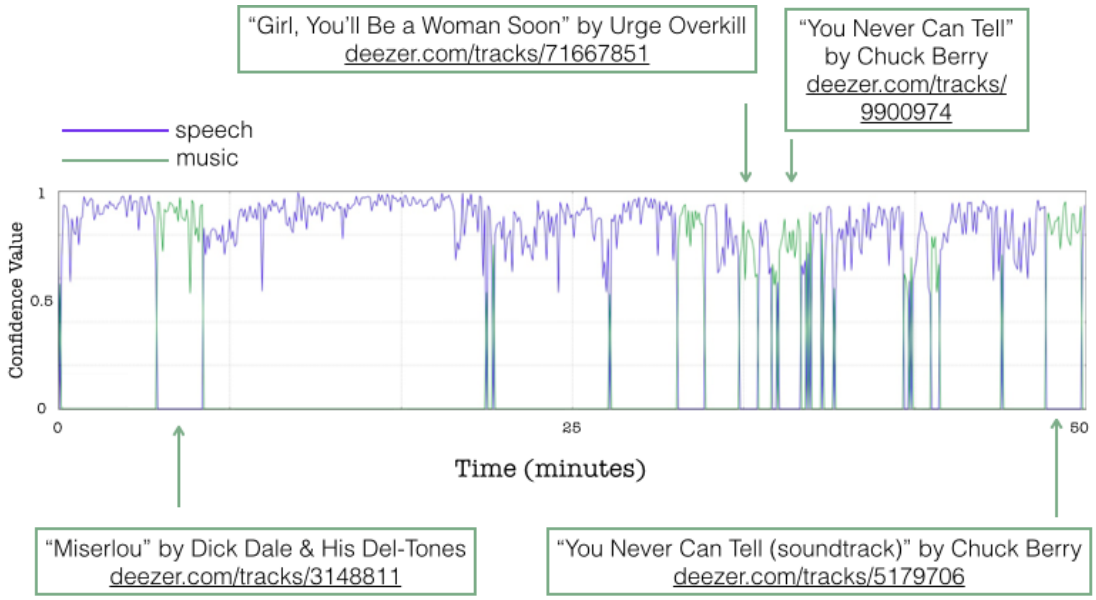


Figure 5.1: Example of song detection in the film "Pulp Fiction".

5.2 Discussion

The goal of this internship was to implement an automatic classification system for audio content using convolutional neural networks.

In a first work phase, we focused on the classification of male vs female speaker voice, setting up a computational system to address this task. We tested the performance of our system, validating its implementation thanks to the good results obtained. Then, we dealt with the classification of speech vs music audio content. We conducted several experiments on a small dataset to study the properties of our system, regarding both the input features and the architecture of the network. We compared the influence of choosing different input features with the same CNN architecture, showing that –consistently with the state for art– low level representations of audio such as the STFT or the CQT perform at least as well as other more complex features, for example MFFC + CHROMA + Δ-MFFC. We have also compared our system with standard classifiers (random forest and support vector machines), obtaining good performance with respect to those. We have put in evidence some expected properties of neural networks, such as the importance of depth and regularization.

The former experimental trails, together with the correct functioning of the system in terms of training time and classification results allowed us to go further. We thus tested our system with larger databases. We obtained good performances on these, showing the scalability of our system. We finally studied the properties of the CNN, showing that structured filters are obtained when adding regularization, and giving some insights about the information that the network preserves.

We remark that a major property of our classification system is that we can use it for multiple classification task without modifying its general architecture. This is clearly a significant advantage with respect to other classifiers that need specific input features designed to each particular task. From a practical point of view, even if we are not able to explicitly say what "the system learned", this generality of our system is clearly a benefit. This, of course, does not mean that we should not try to understand the behavior of our network for each spe-

cific task and try to make progress in theoretical aspects of CNNs applied to audio related tasks.

In conclusion, we have succeeded in implementing a performant and scalable classification system that present good structural properties. Moreover, this system will be used in real-scale industrial applications, related to the improvement of Deezer's database content qualification system.

5.3 Future Work

Thanks to the results obtain with our classification system in the speech vs music classification problem, we hope to extend this approach to address other classification tasks. For example, we could intend to scale the male vs female speech voice task to more general vocal track and singer problems, such as gender and language detection. This of course represents a harder task, so we should consider a more complex processing of the input data, for example the application of source separation techniques or harmonic parts enhancing. In general, we hope to use the implemented system in any classification task that could ameliorate Deezer database tag system, such as instruments detection in musical tracks.

From a practical point of view, when dealing with more complex classification tasks, we should study the application of more elaborated training methods. Several of those have been developed in the literature, especially with respect to the gradient descent processes. Also, we should consider more intelligent initialization of filters depending on the classification task and the augmentation of the data available for training.

Regarding more theoretical considerations, we would like to continue the analysis of CNNs applied to audio related tasks. In particular, we hope to better understand what the system learn. In this line, there are several method that we could implement to obtain a better comprehension of the networks behavior, for example, the visualization of maximally activated maps or the occlusion of selected areas of input features. Moreover, we would like to test the performance of the classifier under natural transformations of audio such as different equalizations, transpositions, or others.

List of Tables

List of Figures

Bibliography

[All05] G. Allaire. *Analyse numérique et optimisation : Une introduction à la modélisation mathématique et à la simulation numérique*. Ecole Polytechnique, 2005.

[AM11] Joakim Andén and Stéphane Mallat. Multiscale scattering for audio classification. In *Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011, Miami, Florida, USA, October 24-28, 2011*, pages 657–662, 2011.

[AM14] Joakim Andén and Stéphane Mallat. Deep scattering spectrum. *IEEE Transactions on Signal Processing*, 62(16):4114–4128, 2014.

[BBB+10] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 3 – 10, 2010.

[Ben09] Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009.

[BGC15] Yoshua Bengio, Ian J. Goodfellow, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2015.

[Bot] L. Bottou. Stochastic gradient descent tricks. http://cilvr.cs.nyu.edu/diglib/lsml/bottou-sgd-tricks-2012.pdf.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[Bro91] J. C. Brown. Calculation of a constant Q spectral transform. *Journal of the Acoustical Society of America,*, 89(1):425–434, 1991.

[CHM+15] Anna Choromanska, Mikael Henaff, Michaël Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*, 2015.

[Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, December 1989.

[DS13] Sander Dieleman and Benjamin Schrauwen. Multiscale approaches to music audio feature learning. In *Proceedings of the 14th International Society for Music Information Retrieval Conference*, November 4-8 2013. http://www.ppgia.pucpr.br/ismir2013/wp-content/uploads/2013/09/69_Paper.pdf.

[DS14] S. Dieleman and B. Schrauwen. End-to-end learning for music audio. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6964–6968, May 2014.

[FDGM86] William M. Fisher, George R. Doddington, and Kathleen M. Goudie-Marshall. The DARPA speech recognition research database: Specifications and status. In *Proceedings of DARPA Workshop on Speech Recognition*, pages 93–99, 1986.

[GL84] D. Griffin and J. Lim. Signal estimation from modified short-time Fourier transform. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 32(2):236–243, 1984.

[HBL12] Eric J. Humphrey, Juan P. Bello, and Yann Lecun. Moving beyond feature design: Deep architectures and automatic feature learning in music informatics. In *Proceedings of the 13th International Society for Music Information Retrieval Conference*, Porto, Portugal, October 8-12 2012. http://ismir2012.ismir.net/event/papers/403-ismir-2012.pdf.

[HDY+12] Geoffrey Hinton, Li Deng, Dong Yu, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath George Dahl, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29(6):82–97, November 2012.

[HE10] Philippe Hamel and Douglas Eck. Learning features from music audio with deep belief networks. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pages 339–344, August 2010.

[Hen11] R. Hennequin. *Décomposition de spectrogrammes musicaux informé par des modèles de synthèse spectrale*. PhD thesis, Telecom ParisTech, 2011.

[HJ10] Chao-Ling Hsu and J.-S.R. Jang. On the improvement of singing voice separation for monaural recordings using the mir-1k dataset. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18(2):310–319, Feb 2010.

[JHTW13] G. James, T. Hastie, R. Tibshirani, and Daniela Witten. *An Introduction to Statistical Learning: With Applications in R*. Springer-Verlag New York Inc, 2013.

[JYL+09] Dalwon Jang, C.D. Yoo, Sunil Lee, Sungwoong Kim, and T. Kalker. Pairwise boosted audio fingerprint. *Information Forensics and Security, IEEE Transactions on*, 4(4):995–1004, Dec 2009.

[Kar] A. Karpathy. Interactively classify toy 2-d data with a neural network. convnetjs deep learning in your browser website. http://cs.stanford.edu/people/karpathy/convnetjs/.

[KB04] J. Kominek and A. W. Black. The cmu arctic speech databases. In *Proc. 5th ISCA Speech Synthesis Workshop*, pages 223–224, 2004.

[KHH+11] Malte Kob, Nathalie Henrich, Hanspeter Herzel, David Howard, Isao Tokuda, and Joe Wolfe. Analysing and Understanding the Singing Voice: Recent Progress and Open Questions. *Current Bioinformatics*, 6(3):1574–8936, 2011.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[KT09] Matthieu Kowalski and Bruno Torrésani. Décompositions parcimonieuses et persistantes de signaux multicanaux. Applications aux signaux MEEG. In *CAp 2008*, pages pages 105–120, Porquerolles, France, May 2009.

[LPLN09] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In Y. Bengio, D. Schuurmans, J.D. Lafferty, C.K.I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 1096–1104. Curran Associates, Inc., 2009.

[LWM+09] E. Law, K. West, M. Mandel, M. Bay, and J. S. Downie. Evaluation of algorithms using games: the case of music annotation. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR)*, 2009.

[MEF+10] Benoit Mathieu, Slim Essid, Thomas Fillon, Jacques Prado, and Gaël Richard. Yaafe, an easy to use and efficient audio feature extraction software. In *Proceedings of the 11th International Society for Music Information Retrieval Conference*, pages 441–446, Utrecht, The Netherlands, August 9-13 2010. http://ismir2010.ismir.net/proceedings/ismir2010-75.pdf.

[Nie] Michael Nielsen. Neural networks and deep learning web resource. http://neuralnetworksanddeeplearning.com/.

[NTR+14] Shrikanth Narayanan, Asterios Toutios, Vikram Ramanarayanan, Adam Lammert, Jangwon Kim, Sungbok Lee, Krishna Nayak, Yoon-Chul Kim, Yinghua Zhu, Louis Goldstein, Dani Byrd, Erik Bresch, Prasanta Ghosh, Athanasios Katsamanis, and Michael Proctor. Real-time magnetic resonance imaging and electromagnetic articulography database for speech production research (tc). *The Journal of the Acoustical Society of America*, 136(3):1307–1311, 2014.

[Ola] C. Olah. Neural networks, manifolds, and topology. web resource. http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/.

[Pee04] G. Peeters. A large set of audio features for sound description (similarity and classification) in the CUIDADO project. Tech. rep., IRCAM, 2004.

[Pra10] J. Prado. Transformée à Q constant. Technical report, TelecomParistech, 2010.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[SB14] J. Schluter and S. Bock. Improved musical onset detection with convolutional neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6979–6983, May 2014.

[Sch98] Eric D. Scheirer. Tempo and beat analysis of acoustic musical signals. *Journal of the Acoustical Society of America*, 103(1):588–601, 1998.

[SKW+10] B. Schuller, C. Kozielski, F. Weninger, F. Eyben, and G. Rigoll. Vocalist gender recognition in recorded popular music. In *Proceedings of the 11th International Society for Music Information Retrieval Conference, ISMIR 2010, Utrecht, Netherlands, August 9-13*. International Society for Music Information Retrieval, 2010.

[SS04] Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression, 2004.

[TC01] G. Tzanetakis and P. Cook. Automatic musical genre classification of audio signals. In *Proceedings of the 2nd International Conference on Music Information Retrieval (ISMIR 2001), Bloomington, Indiana*, 2001.

[Tza99] G. Tzanetakis. Gtzan musicspeech. Availabe online at http://marsyas.info/download/data sets/, 1999.

[WWS11] Felix Weninger, Martin Wöllmer, and Björn Schuller. Automatic assessment of singer traits in popular music: Gender, age, height and race. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, pages 37–42, Miami (Florida), USA, October 24-28 2011. http://ismir2011.ismir.net/papers/OS1-4.pdf.

[ZF14] MatthewD. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, volume 8689 of *Lecture Notes in Computer Science*, pages 818–833. Springer International Publishing, 2014.

[ZRM+13] M.D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q.V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G.E. Hinton. On rectified linear units for speech processing. In *38th International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Vancouver, 2013.

[ZTF11] M.D. Zeiler, G.W. Taylor, and R. Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2018–2025, Nov 2011.