Deep neural networks for music and audio tagging

Jordi Pons

TESI DOCTORAL UPF / 2019

Director de la tesi:

Dr. Xavier Serra i Casals
Dept. of Information and Communication Technologies
Universitat Pompeu Fabra, Barcelona

Dissertation submitted to the Deptartment of Information and Communication Technologies of Universitat Pompeu Fabra in partial fulfillment of the requirements for the degree of:

DOCTOR PER LA UNIVERSITAT POMPEU FABRA

Music Technology Group, Department of Information and Communication Technologies, Universitat Pompeu Fabra, Barcelona.

A l'Alba, a la mare i a l'àvia.

A tres generacions de dones
que sempre m'han fet costat.

This thesis is the result of the work done from September 2015 to September 2019 under the supervision of Xavier Serra at the Music Technology Group (DTIC, Universitat Pompeu Fabra, Barcelona). This work was partially funded by the Maria de Maeztu Strategic Research Program of the Department of Information and Communication Technologies (DTIC) at UPF (MDM-2015-0502), and we are also grateful to NVidia for the donated GPUs. Part of this work was developed in Pandora Radio (Chapter 5), and in Telefónica Research (Chapter 6) during the internships that Jordi carried out in Summer 2017 and Summer 2018, respectively.

Acknowledgments

Moltes (moltíssimes!) gràcies a tota la família. Tot i ser del tipus de persones que *"estudia fins als 30"* i que ha decidit dedicar-se al camp de la música.. tampoc he sortit tant malament, no? Seré doctor! Moltes gràcies per estar sempre al meu costat, i per ajudar-me a fer aquest somni realitat.

A l'Alba, per ajudar-me a mantenir el pols a un dia a dia ple de frustracions. Donat que la majoria d'idees acaben descartades, el simple goig de celebrar-ne una de bona fa que molts continuem fent investigació. Gràcies pel suport quan res funciona! I, sobretot, per celebrar amb mi les petites victòries.

Sincere thanks to Axel Roebel, who was my first mentor and from whom I've learnt how to think scientifically. I will always remember that during my first week working at IRCAM (while being a young and inexperienced undergraduate) he came to my office for a discussion, and he rapidly left saying: *"you have no idea!"*. Hopefully, after defending this thesis, I will be able to show that *"I now have some ideas!"* :) After a very stimulating research stay with him and Marco Liuni, I was determined to pursue a PhD. And here I am, writing the acknowledgments of my thesis!

Una altra persona fonamental al llarg de la meva carrera ha estat en Xavier Serra. Primer, gràcies per fundar i dirigir l'MTG — una

institució que ha esdevingut un referent mundial (a Barcelona!) en la investigació de tecnologies musicals. I segon, gràcies per donar-me l'espai i la confiança necessaris per desenvolupar la meva pròpia investigació. Gràcies! Sense tu, aquesta tesi no hauria estat possible.

I also want to acknowledge the tremendous influence my mentors and close collaborators had in me. Infinite thanks to Oriol Nieto: for your friendship, inspiration, and to teach me not to be scared of coding. To Joan Serrà: with our short collaboration, you stimulated me to do more innovative deep learning research. To Xavier Favory: for our friendship and to teach me the *"savoir faire"* of a good researcher. And to Minz Won: for challenging me all along with our discussions. Thank you all for helping me become a better researcher!

It was also very fruitful to collaborate with Francesc Lluís and with Dario Rethage, two bright master students who helped us with our research project on end-to-end music source separation. Thanks!

Infinite thanks to all the MTG family, as well: Rong, Olga, Marius, Pritish, Rafa, Albin, Pablo, Oriol Romaní, Dmitry, Panos, António Ramires, Sergio Oramas, Andrés Ferraro, Alastair, Frederic, Sonia, Cristina, Merlijn, Emilia, Jordi Bonada, Jordi Janer, Rafael Ramírez, Sergi Jordà, Ángel Faraldo, Julio Carabias, Julián Urbano, Juanjo, Álvaro, Juan, Lorenzo, Helena, Furkan, Emir, Eduardo, Philip, Ajay, Sankalp, Sertan, Sergio Giraldo, David, Fabio, and Angel Blanco.

Also thanks to all the people who contributed, in one way or another, to improve the quality of this manuscript: JP Lewis, Peter M. Todd, Meinard Müller, and (special thanks to) Perfecto Herrera.

Abstract

ENGLISH — Automatic music and audio tagging can help increase the retrieval and re-use possibilities of many audio databases that remain poorly labeled. In this dissertation, we tackle the task of music and audio tagging from the deep learning perspective and, within that context, we address the following research questions:

(i) Which deep learning architectures are most appropriate for (music) audio signals?

(ii) In which scenarios is waveform-based end-to-end learning feasible?

(iii) How much data is required for carrying out competitive deep learning research?

In pursuit of answering research question *(i)*, we propose to use musically motivated convolutional neural networks as an alternative to designing deep learning models that is based on domain knowledge, and we evaluate several deep learning architectures for audio at a low computational cost with a novel methodology based on non-trained (randomly weighted) convolutional neural networks. Throughout our work, we find that employing music and audio domain knowledge during the model's design can help improve the efficiency, interpretability, and performance of spectrogram-based deep learning models.

For research questions *(ii)* and *(iii)*, we perform a study with the SampleCNN, a recently proposed end-to-end learning model, to assess its viability for music audio tagging when variable amounts of training data —ranging from 25k to 1.2M songs— are available. We compare the SampleCNN against a spectrogram-based architecture that is musically motivated and conclude that, given enough data, end-to-end learning models can achieve better results.

Finally, throughout our quest for answering research question *(iii)*, we also investigate whether a naive regularization of the solution space, prototypical networks, transfer learning, or their combination, can foster deep learning models to better leverage a small number of training examples. Results indicate that transfer learning and prototypical networks are powerful strategies in such low-data regimes.

CATALÀ — L'etiquetatge automàtic d'àudio i de música pot augmentar les possibilitats de reutilització de moltes de les bases de dades d'àudio que romanen pràcticament sense etiquetar. En aquesta tesi, abordem la tasca de l'etiquetatge automàtic d'àudio i de música des de la perspectiva de l'aprenentatge profund i, en aquest context, abordem les següents qüestions científiques:

(i) Quines arquitectures d'aprenentatge profund són les més adients per a senyals d'àudio (musicals)?

(ii) En quins escenaris és viable que els models d'aprenentatge profund processin directament formes d'ona?

(iii) Quantes dades es necessiten per dur a terme estudis d'investigació en aprenentatge profund?

Per tal de respondre a la primera pregunta *(i)*, proposem utilitzar xarxes neuronals convolucionals motivades musicalment i avaluem

diverses arquitectures d'aprenentatge profund per a àudio a un baix cost computacional. Al llarg de les nostres investigacions, trobem que els coneixements previs que tenim sobre la música i l'àudio ens poden ajudar a millorar l'eficiència, la interpretabilitat i el rendiment dels models d'aprenentatge basats en espectrogrames.

Per a les preguntes *(ii – iii)* estudiem com el SampleCNN, un model d'aprenentatge profund que processa formes d'ona, funciona quan disposem de quantitats variables de dades d'entrenament — des de 25k cançons fins a 1'2M cançons. En aquest estudi, comparem el SampleCNN amb una arquitectura basada en espectrogrames que està motivada musicalment. Els resultats experimentals que obtenim indiquen que, en escenaris on disposem de suficients dades, els models d'aprenentatge profund que processen formes d'ona (com el SampleCNN) poden aconseguir millors resultats que els que processen espectrogrames.

Finalment, per tal d'intentar respondre a la pregunta *(iii)*, també investiguem si una regularització severa de l'espai de solucions, les xarxes prototipades, l'aprenentatge per transferència de coneixement, o la seva combinació, poden permetre als models d'aprenentatge profund obtenir més bons resultats en escenaris on no hi ha gaires dades d'entrenament. Els resultats dels nostres experiments indiquen que l'aprenentatge per transferència de coneixement i les xarxes prototipades són estratègies útils quan les dades d'entrenament no són abundants.

Contents

List of Figures

List of Tables

List of Abbreviations

SGD Stochastic Gradient Descent

MLP Multi Layer Perceptron

CNN Convolutional Neural Network

RNN Recurrent Neural Network

BN Batch Normalization

MP Max-pool layer

ReLU Rectified Linear Unit

STFT Short-Time Fourier Transform

SVM Support Vector Machine

ELM Extreme Learning Machine

MSD Million Song Dataset

MTT MagnaTagATune dataset

US8K UrbanSound8K dataset

ASC-TUT Acoustic Scene Classification dataset

Chapter 1

Motivation and scope

The amount of music and audio recordings that are accessible is constantly growing. However, most of these recordings are poorly labeled and this difficult its identification and access. Indexing such audio content with semantic labels has been a research topic for the past two decades, because having such semantic information per audio or music track would allow to better organize the existing audio repositories. If solved, auto-tagging would enable users to better explore their audio collections, which would increase the retrieval and re-use possibilities of this data. Hence, automatic audio tagging systems aim to predict semantically relevant tags from the audio signal. This requires extracting acoustic features that are good estimators of the type of tags we are interested in, followed by a single or multi-label classification stage — or in some cases, regression stage (Choi et al., 2016). A conventional approach consists in carefully designing these features considering the domain expertise of a trained researcher (Sordo et al., 2007; Prockup et al., 2015; Bayle et al., 2017). However, it is difficult to envision what features are relevant to the

task at hand. To address this challenge, artificial neural networks have been explored to unify the feature extraction with the machine learning classifier, and recent publications are showing promising results using deep neural networks (Dieleman and Schrauwen, 2014; Choi et al., 2016; Lee et al., 2018). This dissertation focuses on using deep neural networks for automatically tagging music and audio content.

This manuscript starts introducing the basic building blocks of deep artificial neural networks (in Chapter 2), to later discuss how these have been historically used for music informatics. Along with this overview, we also discuss several use cases to help the reader understand the possibilities (and limitations) of current artificial neural networks for music and audio. Out of this review, we found that:

(i) Deep artificial neural networks are a suitable tool for modeling music computationally.

(ii) It exists an "end-to-end learning trend" among deep learning researchers, who are exploring the possibilities of this approach.

(iii) Artificial neural networks require a significant amount of data to be competitive.

These ideas have greatly influenced the development of this thesis, and understanding their scope can be of utility to comprehend the research directions we decided to pursue. Back when our work started in 2015, the state of the above principles was the following:

(i) Artificial neural networks were not widely used for music and audio. Hence, deep learning was still a promise to be explored and it was not clear how researchers would adopt it.

(ii) "End-to-end learning for audio is an impossible endeavour". It existed the idea that for end-to-end learning to be viable, much more computing power and training data were required.

(iii) Public research institutions (like our university) were severely concerned by the computing power and data resources required to do competitive research on artificial neural networks.

Although the field of artificial neural networks for music and audio has advanced significantly, it is interesting to see how these ideas are still very present. For example, note that *(i)* ongoing research is exploring which deep learning architectures are more suitable for music and audio signals (Ravanelli and Bengio, 2018; Won et al., 2019); *(ii)* recent works are exploring the possibilities of end-to-end learning for (music) audio (Lluís et al., 2018; Stoller et al., 2019); and *(iii)* research institutions are moving forward for building the required infrastructure (data & computing power) for carrying out competitive deep learning research (Fonseca et al., 2017; Stöter et al., 2018). We frame our work in that context, and our aim is to contribute towards answering these research questions that the field is facing:

(i) Which deep learning architectures are most appropriate for (music) audio signals?

(ii) In which scenarios is waveform-based end-to-end learning feasible?

(iii) How much data is required for carrying out competitive deep learning research?

1.1 Scope of our work: audio tagging

Provided that music audio tagging *(i)* is a challenging task due to how variate the target tags can be, and *(ii)* a collection of readily available datasets and established baselines are available, we decided to set the music audio tagging task as our main study case. Besides, we observed that many of the scientific challenges that music audio tagging faces (those are listed in Chapter 2) are also relevant for the general audio tagging case. For that reason, we also explore the general audio tagging setting to study how these technologies behave with different data. Hence, throughout this dissertation two main use cases are studied:

- Music Audio Tagging: these systems aim at predicting musically relevant tags from the audio signal. Examples of those can be *meter tags* (e.g., triple-meter, cut-time), *rhythmic tags* (e.g., swing, syncopation), *harmonic tags* (e.g., major, minor), *mood tags* (e.g., angry, sad), *vocal tags* (e.g., male, female, vocal grittiness), *instrumentation tags* (e.g., piano, guitar), *sonority tags* (e.g., live, acoustic), or *genre tags* (e.g., jazz, rock, disco).

- General Audio Tagging: these systems aim at predicting acoustically relevant tags from the audio signal. Examples of those can be *acoustic events* (e.g., air conditioner, car horn, drilling, gunshot, siren), *animal sounds* (e.g., dog bark, birds chirping, pig oink), *human sounds* (e.g., fart, male/female speech), *acoustic scenes* (e.g., lakeside beach, forest path, metro-station, office), or even *music sounds* (e.g., guitar, trumpet, rock).

1.2 Outline of the thesis

In the following, we provide an overview of what is presented along with this thesis. Our main experiments and contributions can be summarized as follows:

- **We propose to use musically motivated CNNs for music audio tagging**. Within this context, the CNN filters we employ are designed to fit specific musical concepts that are of relevance for the task at hand. Hence, the design strategy we propose heavily relies on musical/audio domain knowledge. With this work, our goal is to investigate which architectures are more appropriate for modeling music signals with deep learning. We found that musically motivated CNNs strongly regularize the solution space. This can be beneficial in scenarios where few training data are available. However, this can over-constrain the solution space in scenarios where sizable amounts of training data are available. Still, for most scenarios, musically motivated CNNs perform similarly to its counterparts while requiring less computational resources and being more interpretable.

  More details on musically motivated CNNs are in Chapter 3.

- **We investigate to use non-trained (randomly weighted) CNNs as feature extractors for audio tagging.** With the goal to compare classification accuracies when using different randomly weighted CNN feature extractors, we use the features extracted from the embeddings of deep architectures as input to a classifier. Given that the performance delivered by random

CNN features is correlated with the results of their end-to-end trained counter-parts (Saxe et al., 2011), we can bypass the time-consuming and hardware-consuming process of training for evaluating a given neural network architecture. By following this methodology, we run a comprehensive evaluation of the current deep architectures for audio tagging.

To know more about the method we employed and the results we obtained, see Chapter 4.

- **We investigate how several music audio tagging deep architectures perform at scale.** We study how different deep architectures perform when trained with datasets of variable size: the MagnaTagATune (25k songs), the Million Song Dataset (240k songs), and a private dataset of 1.2M songs. We compare musically motivated CNNs with SampleCNNs, a waveform-based end-to-end learning model by Lee et al. (2018). These are representative of two opposite design strategies: one heavily relies on domain knowledge while the other does not. Our results suggest that models relying on domain knowledge play a relevant role in scenarios where no sizable datasets are available. However, given enough data, those end-to-end learning models not relying on domain knowledge can achieve better results — possibly because its solution space is not constrained by strong assumptions regarding the nature of the signal or task.

  More details about this study are available in Chapter 5.

- **We investigate how to train neural network-based general audio taggers with few data.** In particular, we study whether *a)* a naive regularization of the solution space, *b)* pro-

totypical networks, *c)* transfer learning, or *d)* their combination, can foster deep learning models to better leverage a small amount of training examples. Results indicate that transfer learning is a powerful strategy in such scenarios. However, transfer learning assumes that a pre-trained model is readily available. Importantly, such model needs to be pre-trained with similar data to the few samples that are accessible for training. When these data do not match, we show that prototypical networks trained from scratch can be the right choice.

To know more about prototypical networks, transfer learning, or for further details about this study, see Chapter 6.

As seen, we centered our efforts in trying to provide answers to these fundamental research questions we identified along with our introductory discussion (that we present in Chapter 2).

Chapter 2

Neural networks for music: an introductory journey through their history

Many feats have happened between the pioneering papers written by J. P. Lewis and Peter M. Todd in the 80's and the current wave of generative adversarial networks' composers. Along that journey, the connectionists' work was forgotten during the artificial intelligence winter, very influential names (like Juergen Schmidhuber, Yann Le-Cun or Andrew Ng) contributed with seminal publications and, in the meantime, researchers have made tons of progress.

The goal of this chapter is to provide an historical introduction to the field of artificial neural networks for music. We won't go through every single paper in the field, but we'll cover what are the milestones that helped to shape the current state-of-the-art. In the following, we will be first introducing the basic building blocks of modern artificial neural networks to later discuss how these have been historically used

for music informatics. Along with that discussion, we will also introduce music use cases to help the reader understand the possibilities (and limitations) of current artificial neural networks for music.

2.1 Introduction to artificial neural networks for music

This section introduces the basic building blocks of modern artificial neural networks for music. Provided that most of these models can be explained as a combination of several well-established architectures, along that section we outline the most used ones: the multilayer perceptron (MLP), convolutional neural networks (CNNs), recurrent neural networks (RNNs, or its LSTMs variant), and attention. The goal of these models is to approximate some function of interest. For example, to build a music genre classifier, such function $\hat{\mathbf{y}} = f(\mathbf{x})$ maps an audio input $\mathbf{x}$ to a genre category $\hat{\mathbf{y}}$. Or, as another example, $\hat{\mathbf{y}} = f(\mathbf{x})$ can map a music mixture $\mathbf{x}$ into a separated singing-voice solo track $\hat{\mathbf{y}}$ to build a singing voice source separation model. In other words, artificial neural networks are powerful models capable to parameterize rather complex mapping functions that we express as $\hat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represent the set of learnable parameters of a model.

The goal of this section is to cover the basics of the field. For that reason, we might intentionally omit some of the models' details since our intention is to guide the reader through the process of learning the set of intuitions required to effectively understand artificial neural networks for music. Additional details can be easily found in the original references that we list along with the text.

In that section, we first introduce the foundations of the learning algorithms used for training artificial neural networks from data, and then we discuss the pros and cons of each architecture. To allow a better understanding on the usability of artificial neural networks for music, we also include practical music examples to further illustrate which are the merits of each architecture. For example, in Section 2.1.8 we discuss the suitability of the introduced architectures when using spectrogram inputs, a highly performant and widely employed input setup (Dieleman and Schrauwen, 2014; Choi et al., 2017a).

2.1.1 Training artificial neural networks

The quintessential algorithm for training neural networks is stochastic gradient descent (SGD). Although this algorithm does not guarantee to reach a global minimum when optimizing non-convex error functions (like it happens to be when training deep neural networks), SGD works well in practice. SGD updates the model parameters' $\boldsymbol{\theta}$ as follows (Robbins and Monro, 1951):

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \mu_i \nabla e(\boldsymbol{\theta}_i). \tag{2.1}$$

The i sub-index denotes the sequential nature of the training algorithm, where the parameters are updated iteratively following the above equations. To start, $\boldsymbol{\theta}_i$ parameters are (randomly) initialized and the backpropagation algorithm (Rumelhart et al., 1986) finds the update direction $\nabla e(\boldsymbol{\theta}_i)$ by means of computing the gradient of the error function with respect to the parameters to optimize. Note that the parameter updates follow the negative direction pointed by the gradient, which is the direction having the steepest descent. Finally,

μ_i (known as the learning rate) is a scalar that controls how much the parameters $\boldsymbol{\theta}_i$ are updated. Although SGD is mostly used for supervised learning[1] —because it is easy to compute the error $e(\mathbf{y}, \hat{\mathbf{y}})$ between the target and the prediction—, SGD can also be used for unsupervised learning.[2] In these scenarios, a common trick is to use the reconstruction error $e(\mathbf{x}, \hat{\mathbf{x}})$ to inform training (Bretan et al., 2017).

To further discuss how to train artificial neural networks, let's consider the music genre classification example introduced above. We want to use this example to remark that SGD is stochastic: it randomly selects a subset of the training data (a batch) to compute the gradient that guides learning. For each update, we will first run the audio examples (a batch) through the model to estimate the corresponding genre labels $\hat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta}_i)$. Considering these estimates, the backpropagation algorithm (Rumelhart et al., 1986) uses the errors $e(\mathbf{y}, \hat{\mathbf{y}})$ to compute the gradients — that are averaged across the batch to compute $\nabla e(\boldsymbol{\theta}_i)$. Once $\nabla e(\boldsymbol{\theta}_i)$ is known, the model parameters' $\boldsymbol{\theta}_i$ can be updated accordingly. Note, then, that SGD updates $\boldsymbol{\theta}_i$ every batch. To conclude, we want to note that it exists several SGD extensions that are widely used. Some iconic ones are the momentum variant (Rumelhart et al., 1986) or adam (Kingma and Ba, 2014), that are based on the same iterative principle as SGD.

After introducing how to train artificial neural networks, we devote the rest of Section 2.1 to present the most popular artificial neural network architectures that can be used to define $\hat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta})$.

[1] Supervised learning aims at learning to predict a target $\mathbf{y}$ given $\mathbf{x}$: $p(\mathbf{y}|\mathbf{x})$.
[2] Unsupervised learning aims to learn the distribution of the data $\mathbf{x}$: $p(\mathbf{x})$.

2.1.2 Multilayer perceptrons

The basis of modern artificial neural networks is the perceptron, that infers a prediction out of the following equation (Rosenblatt, 1957):

$$\hat{\mathbf{y}} = \sigma_{(0)}(\mathbf{W}_{(0)}\mathbf{x} + \mathbf{b}_{(0)}). \tag{2.2}$$

Although the perceptron is a single-layered linear model (with $L = 0$), modern neural networks are generally formed by more than one hidden layer. These deep perceptrons having more than one hidden layer are also known as multilayer perceptrons (MLPs, with $L \geq 1$). For example, for $L = 2$ the MLP is defined as follows:

$$\hat{\mathbf{y}} = \sigma_{(2)}(\mathbf{W}_{(2)}\mathbf{h}_{(2)} + \mathbf{b}_{(2)}) \tag{2.3}$$

$$\mathbf{h}_{(2)} = \sigma_{(1)}(\mathbf{W}_{(1)}\mathbf{h}_{(1)} + \mathbf{b}_{(1)}) \tag{2.4}$$

$$\mathbf{h}_{(1)} = \sigma_{(0)}(\mathbf{W}_{(0)}\mathbf{x} + \mathbf{b}_{(0)}), \tag{2.5}$$

where the predicted output is $\hat{\mathbf{y}} \in \Re^{d_{output}}$, $\sigma_{(l)}$ can be any activation function, the weight matrices are $\mathbf{W}_{(l)} \in \Re^{d_{(l)} \times d_{(l-1)}}$, the bias vector is $\mathbf{b}_{(l)} \in \Re^{d_{(l)}}$, and the input is $\mathbf{x} \in \Re^{d_{input}}$. Latent representations (out of the hidden layers) are in $\mathbf{h}_{(l)}$, and l stands for the index of the current layer (ranging from $0 \leq l \leq L$). Further, the dimension d can be different in each layer, input, and output — that's why $d_{(l)}$ is denoted with the l sub-index. Each layer is characterized by a non-linear activation function $\sigma_{(l)}$, that can be layer specific. Popular activation functions $\sigma_{(l)}$ are the sigmoid, the tanh or the rectified linear unit (ReLU). The above equations assume 1D inputs, but any input signal can be fed to the model by simply reshaping. For example, a 2D spectrogram can be reshaped into a 1D vector.

We have seen that MLPs predict an output via computing a weighted average considering *all* its input, note that $\mathbf{W}_{(l)}$ interacts with the whole input signal (either $\mathbf{x}$ or $\mathbf{h}_{(l)}$). As an example on the usage of MLPs, let's say we aim to predict whether an audio contains music or speech. In this scenario, and assuming that our latent $\mathbf{h}_{(2)}$ representation contains relevant features like timbre and/or loudness, we want to weight with $\mathbf{W}_{(2)}$ the relevance of these $\mathbf{h}_{(2)}$ features to predict ($\hat{\mathbf{y}}$) whether our input contains music or speech. Note that the same rationale also applies to the lower layers of the model, where this weighted average would help defining more discriminative features.

2.1.3 Convolutional neural networks

Convolutional neural networks (CNNs) are simply neural networks that use convolutional operators in place of matrix multiplications in at least one of their layers (LeCun et al., 1989). The discrete 1D convolution operation used in CNNs is defined as follows:

$$s[n] * w[n] = \sum_{m=-\infty}^{\infty} s[m] \cdot w[n-m]. \tag{2.6}$$

And the 2D variant of the convolution is as follows:

$$s[x,y] * w[x,y] = \sum_{m_1=-\infty}^{\infty} \sum_{m_2=-\infty}^{\infty} s[m_1,m_2] \cdot w[x-m_1, y-m_2], \tag{2.7}$$

where $w[n]$ or $w[x,y]$ shifts, and interacts with the signal $s[n]$ or $s[x,y]$ in a multiplicative-add fashion. To better understand the convolution, in the following lines we introduce examples to further discuss its behaviour.

CNNs replace any MLP layer with a convolutional layer:

$$\mathbf{h}_{(l)}^{(k)} = \sigma_{(l)}(\mathbf{W}_{(l)}^{(k)} * \mathbf{h}_{(l-1)} + \mathbf{b}_{(l)}). \tag{2.8}$$

Or, in case where a CNN is used as input layer (the first layer), the above equation translates to:

$$\mathbf{h}_{(1)}^{(k)} = \sigma_{(0)}(\mathbf{W}_{(0)}^{(k)} * \mathbf{x} + \mathbf{b}_{(0)}). \tag{2.9}$$

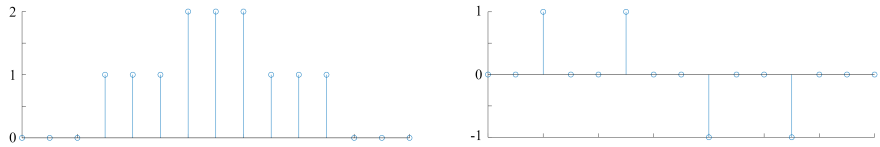
Where the convolutional operator is capable to deal with 1D or 2D signals, and each of the k CNN filters $\mathbf{W}_{(l)}^{(k)}$ capture a local context. For example: if we consider a 2D spectrogram input $\mathbf{x} \in \Re^{T \times F}$, the filters $\mathbf{W}_{(0)}^{(k)} \in \Re^{i \times j}$ will model a local context of $i \times j$ (where $i \leq T$ and $j \leq F$).[3] Note that a MLP can be implemented with a CNN having filters of the following shape: $i = T$ and $j = F$. As seen, each CNN filter defines a local feature. Consequently, out of a CNN layer with k filters, one has k feature maps $\mathbf{h}_{(l)}^{(k)}$ that preserve *locality*.

To further understand this behavior, let us consider a basic 1D convolution example. For simplicity, we will omit the non-linearity $\sigma_{(0)}$ and the bias term $\mathbf{b}_{(0)}$ to focus on the core convolutional operation: $\mathbf{W}_{(0)}^{(k)} * \mathbf{x} \approx \mathbf{h}_{(1)}$. In our example, we will make use of a single filter of length two:

$$\mathbf{W}_{(0)}^{(k=0)} = [1, -1]. \tag{2.10}$$

However, in practice, the filter weights' $\mathbf{W}_{(l)}^{(k)}$ won't be hard-coded but will be learned. Since we utilize a single CNN filter, the output will be composed of a single feature map — see Figure 2.1.

[3] T and F stand for the number of *time frames* and *frequency bins* in a spectrogram, respectively.



(a) 1D input signal $\mathbf{x}$. **(b)** 1D feature map $\approx \mathbf{h}_{(1)}$.

Figure 2.1: Simplification of the CNN model to understand its intuition. We set $\mathbf{h}_{(1)} \approx \mathbf{W}_{(0)}^{(k)} * \mathbf{x}$, and $\mathbf{W}_{(0)}^{(k=0)} = [1, -1]$.

The filter $\mathbf{W}_{(0)}^{(k=0)} = [1, -1]$ shifts along the temporal-axis to compute the difference between two consecutive samples. In other words: for every filter shift, we multiply the two consecutive samples of interest by 1 and -1 to later add those. Consequently, the output is zero if these two consecutive samples hold the same value. Otherwise, the feature map will reflect the magnitude of their difference. In our example one can see that, differently from MLPs, CNNs preserve *locality*: the feature map displays where these differences occur.

Given the nature of the convolution operation, that shifts CNN filters throughout the signal, these layers are particularly useful to encode representations that are time-invariant (in 1D signals like waveforms) or time-frequency invariant (in 2D signals like spectrograms). For that reason, CNNs tend to be used in lower layers of the model[4] because these are powerful at encoding relevant local stationarities — like sinusoids in waveforms (Dieleman and Schrauwen, 2014; Lee et al., 2018), or timbral traces in spectrograms (Pons et al., 2017b).

Finally, this example also showcases the importance of the shape of CNN filters. In our rather simplistic example, the filter only considers a context of two data points to inform its output. This local context is arbitrarily defined, according to our goal, when defining

[4]Including the input layer.

the shape of the CNN filter: $i \times j$. For example, a historical setup for waveform inputs has been to set such context to 512×1 (length = 512), similarly to the way we design an analysis window for the STFT (Dieleman and Schrauwen, 2014).

2.1.4 Recurrent neural networks

Arguably, temporal dependencies (at different time scales) are of importance when modeling music computationally. Recurrent neural networks (RNNs) seem a reasonable fit towards modeling those, since RNNs can explicitly encode temporal dependencies (Elman, 1990). That is because its latent representation $\mathbf{h}_{(1)}^{(t)}$ does not only depend on the current time-step t: $\mathbf{h}_{(1)}^{(t)} = f(\mathbf{x}^{(t)})$, but also depends on the previous time-step: $\mathbf{h}_{(1)}^{(t)} = f(\mathbf{h}_{(1)}^{(t-1)}, \mathbf{x}^{(t)})$. For that reason, RNNs have been historically used to model time-series. The traditional formulation of RNNs (known as vanilla RNNs) is as follows:

$$\hat{\mathbf{y}}^{(t)} = \sigma_{(1)}(\mathbf{W}_{(1)}\mathbf{h}_{(1)}^{(t)} + \mathbf{b}_{(1)}) \tag{2.11}$$

$$\mathbf{h}_{(1)}^{(t)} = \sigma_{(0)}(\mathbf{W}_{(0)}\mathbf{x}^{(t)} + \mathbf{W}_{rec}\mathbf{h}_{(1)}^{(t-1)} + \mathbf{b}_{(0)}), \tag{2.12}$$

where $\mathbf{W}_{rec} \in \Re^{d_{(l)} \times d_{(l)}}$ denotes the recurrent weights that explicitly encode temporal dependencies. Note that $\hat{\mathbf{y}}^{(t)}$ remains the same as in Eq. 2.3, only $\mathbf{h}_{(l)}^{(t)}$ changes to be recursive. Although for this example we only considered a single-layered RNN ($L = 1$), one can stack several RNN layers ($L > 1$) that can even be set to encode opposite temporal directions. For example, a bi-derectional RNN considers representations from the past $\mathbf{h}^{(t-1)}$ and from the future $\mathbf{h}^{(t+1)}$ (Schuster and Paliwal, 1997).

Note that adding the recursive connection $\mathbf{W}_{rec}$ seems a promising step towards modeling the long-term temporal dependencies that are so relevant in music. However, this goal is still far beyond our reach. Unfortunately, RNNs have difficulties in learning long-term dependencies due to the "vanishing/exploding gradient" problem (Bengio et al., 1994). Although it is out of our scope to deeply discuss this phenomenon, to follow our discussion it will suffice to know that the long-term dependencies are hardly reachable at time t because latent representations at time $t-n$ are only accessible via a problematic path defined by $\mathbf{W}_{rec}$. See in Eq. 2.12 that to access $\mathbf{h}^{(t-n)}$ at time t one needs to recursively go through $\mathbf{W}_{rec}$ in a multiplicative fashion for n steps: $\approx {\mathbf{W}_{rec}}^{n}\mathbf{h}^{(t-n)}$. Consequently, if $\mathbf{W}_{rec}$ values are too small: ${\mathbf{W}_{rec}}^{n}\mathbf{h}^{(t-n)}$ will vanish. And, if $\mathbf{W}_{rec}$ values are too big: ${\mathbf{W}_{rec}}^{n}\mathbf{h}^{(t-n)}$ will explode. Although for this didactic simplification we assume that the learning phase already occurred, this problem actually takes place during training: the gradients used for guiding the learning are dominated by ${\mathbf{W}_{rec}}^{n}$ and these can vanish or explode (Goodfellow et al., 2016).

2.1.5 Improving recurrent neural networks: LSTMs

In short, RNNs cannot learn long-term dependencies because these access to past information via a problematic path defined by $\mathbf{W}_{rec}$. A practical solution to this problem is to allow *direct paths* from the past. LSTMs, a RNN variant, explored this solution via defining a state representation ($\mathbf{s}^{(t)}$) that explicitly carries information from the past ($\mathbf{s}^{(t-1)}$).[5]

[5] $\mathbf{s}^{(t)}$ and $\mathbf{s}^{(t-1)}$ are defined in the following page.

LSTMs are defined as follows (Hochreiter and Schmidhuber, 1997):

$$Input: \quad \mathbf{i}^{(t)} = \sigma(\mathbf{W}\mathbf{x}^{(t)} + \mathbf{W}_{rec}\mathbf{h}^{(t-1)} + \mathbf{b}) \tag{2.13}$$

$$State: \quad \mathbf{s}^{(t)} = \mathbf{g}_i^{(t)}\mathbf{i}^{(t)} + \mathbf{g}_f^{(t)}\mathbf{s}^{(t-1)} \tag{2.14}$$

$$Output: \quad \mathbf{h}^{(t)} = \sigma(\mathbf{s}^{(t)})\mathbf{g}_o^{(t)} \tag{2.15}$$

While the path to the past coming from $\mathbf{W}_{rec}\mathbf{h}^{(t-1)}$ (as defined in $\mathbf{i}^{(t)}$) is again multiplied by $\mathbf{W}_{rec}$, the other path to the past $\mathbf{g}_f^{(t)}\mathbf{s}^{(t-1)}$ (as defined in $\mathbf{s}^{(t)}$) is directly accessible — one just needs to bypass a sigmoidal gate $\mathbf{g}_f^{(t)}$. We omitted the layer sub-indices l for clarity, and the $\sigma(\cdot)$'s in Eq. 2.13 and 2.15 can be any non-linearity, but the $tanh(\cdot)$ is oftentimes used (Hochreiter and Schmidhuber, 1997).

Note that the *input* representation formulation $\mathbf{i}^{(t)}$ is the same as the traditional RNN; the *state* representation decides what's important to keep: if "present" ($\mathbf{i}^{(t)}$) or "past" information ($\mathbf{s}^{(t-1)}$); and the *output* representation is to control what's visible from the output. This behavior is controlled by a set of sigmoidal gates: $\mathbf{g}_i^{(t)}$ (that *reads* the input), $\mathbf{g}_f^{(t)}$ (that *forgets* from the past), and $\mathbf{g}_o^{(t)}$ (that *writes* to the output), that are defined as follows:

$$Gates: \quad \mathbf{g}_?^{(t)} = \sigma(\mathbf{b}_? + \mathbf{W}_?\mathbf{x}^{(t)} + \mathbf{W}_{rec_?}\mathbf{h}^{(t-1)}), \tag{2.16}$$

where the ? sub-index is a placeholder for i (standing for input), f (standing for forget), and o (standing for output). Note, then, that each gate has independent parameters (*i.e.*, $W_i \neq W_f \neq W_o$).

Finally, it is important to remark that LSTMs (or similar gated recurrent models like the GRUs by Cho et al. (2014)) are the RNNs that most practitioners use because these actually work in practice.

2.1.6 Attention-based artificial neural networks

An alternative approach to model temporal relationships in music signals is to pay attention to different time positions. The attention idea is again based on allowing *direct paths* from other (potentially distant) time-steps. In this case, these representations are obtained via a weighted average (through time) that considers information from any time position (Bahdanau et al., 2014):

$$\mathbf{h}_{(l)}^{(t)} = \sum_{n=0}^{T} \alpha_{(t,n)} \mathbf{h}_{(l-1)}^{(n)}, \tag{2.17}$$

thus meaning that at time t one can directly access to any $\mathbf{h}_{(l-1)}^{(n)}$, where $n \in [0, T]$ and T is an arbitrarily defined time horizon. Note that each $\mathbf{h}_{(l-1)}^{(n)}$ is weighted by an $\alpha_{(t,n)}$ matrix with time independent attention weights (normalized through time by a softmax):

$$\alpha_{(t,n)} = softmax\Big(b_{(t,n)}\Big) = \frac{\mathrm{e}^{b_{(t,n)}}}{\sum_{k=0}^{T} \mathrm{e}^{b_{(t,k)}}}, \tag{2.18}$$

and the $b_{(t,n)}$'s are dynamically estimated by a function $f(\cdot)$ that considers some context $c_{(t,n)}$ to inform its prediction:

$$b_{(t,n)} = f(c_{(t,n)}). \tag{2.19}$$

For example, such context could be set to: $f(\mathbf{h}_{(l-1)}^{(t)}, \mathbf{h}_{(l-1)}^{(n)})$. This meaning that the amount of attention $\alpha_{(t,n)}$ allocated at time t would be estimated considering the information available at time t and at time n. However, one can define this context in manifold ways (Raffel and Ellis, 2016; Raffel, 2016). Furthermore, ongoing research is also looking at different ways to set $f(\cdot)$, e.g., it could be set to be any

operation like the dot product or it could be learnt (Bahdanau et al., 2014; Huang et al., 2019; Raffel, 2016).

As seen, we defined the $\alpha_{(t,n)}$ attention weights so that they dynamically depend on the actual content that is being analyzed — this kind of attention is also known as self-attention. In short, self-attention models relate different positions of the *same* input sequence. Provided that all the attention-based models for music published till date utilize self-attention, throughout this article we will indistinctly use the terms attention and self-attention (Raffel, 2016; Huang et al., 2019). Although alternative attention mechanisms could be used (Bahdanau et al., 2014), the attention models we discuss are designed to dynamically pay attention (with larger $\alpha_{(t,n)}$ weights) to the most important parts of its input.

2.1.7 Attention vs. RNN

As we will see in Section 2.3, RNNs and attention-based artificial neural networks have been vigorously used for algorithmic music composition (Todd, 1988; Eck and Schmidhuber, 2002; Huang et al., 2019). The most common way to utilize those models for composing music is to set them as single-step predictors: they learn to predict notes at time $t+1$ using earlier notes (at times $\leq t$) as inputs. After training has finished, those models are primed with an input music seed to generate a new output. Each output is sequentially fed back as input to the model for generating consecutive outputs in an autoregressive fashion. In order to facilitate the task of composition, these models directly operate over scores. The symbolic music data that is often used in algorithmic music composition (e.g., MIDI scores) is generally represented as a sequence of discrete tokens (e.g., one-hot vectors) that encode the notes in a suitable format for neural networks.

The algorithmic music composition case we described above is particularly interesting when comparing attention-based models vs. RNNs, because a music piece generally contains local motives and global structure. For example: a compelling song might incorporate local motives that might be repeated in different sections, which creates a sense of coherence and global structure. Having that in mind, which artificial neural network seems more appropriate for the algorithmic music composition task? Differently from RNNs (or its LSTMs variant), attention-based artificial neural networks have the capacity to capture long-term dependencies by design. While attention-based models can arbitrarily access to any time position in the past and/or in the future to inform a decision, RNNs or LSTMs cannot. Although LSTMs provide a *direct path* to a stored memory state, this direct temporal link cannot guarantee the access to any (potentially distant) time-position. At this point, it might be clarifying to reveal what LSTM means: Long Short-Term Memory. Hence, the name itself reflects the limits of this architecture, that it can only learn short-term dependencies that are *long*. Provided that long-term dependencies are of relevance in music, it seems a natural fit to utilize attention-based models to capture those with neural networks. By following a similar rationale, the algorithmic music composition literature arrived to equivalent conclusions that align with our discussion. The early approaches were using vanilla RNNs (Todd, 1988). Later, LSTMs were introduced as a way to improve the global coherence of the generated pieces (Eck and Schmidhuber, 2002), and current state-of-the-art models are exploring to use attention-based artificial neural networks (Huang et al., 2019).

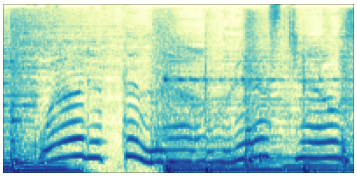
2.1.8 Sectrogram-based artificial neural networks

The building blocks of modern deep learning architectures are the multilayer perceptron (MLP), convolutional neural networks (CNNs), recurrent neural networks (RNNs), and attention. We have seen that MLPs compute a prediction considering *all* its input, that CNNs preserve *locality* by computing filter-selective feature maps, that RNNs can capture *short-term* temporal dependencies, and that attention can capture *long-term* temporal dependencies. In Figure 2.2 we exemplify all these intuitions graphically with a spectrogram example. Provided that spectrogram inputs are widely used due to their good performance in several music tasks of relevance —including auto-tagging (Pons et al., 2018), source separation (Jansson et al., 2017) or transcription (Southall et al., 2017)—, we introduce this discussion to guide the reader through the process of designing a state-of-the-art artificial neural network.

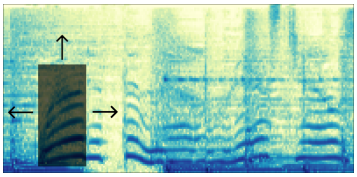
Before all else, spectrograms need to be pre-processed and normalized so that neural networks can deliver a good performance. A common setup consists of using log-mel spectrograms, and to normalize those to be zero-mean and unit-var (Choi et al., 2018b). The mel mapping reduces the size of the input by providing less resolution to the perceptually irrelevant parts of the spectrogram (Stevens et al., 1937), and the logarithmic compression reduces the dynamic range of the input. Finally, the zero-mean and unit-var normalization centers the data around zero, which facilitates process (Glorot and Bengio, 2010).

When a MLP is employed, *all* its input is used to compute a weighted average output. This means that for every output one needs to learn as many weights as spectrogram bins are visible in the input (see equations in Section 2.1.2). Consequently, for high

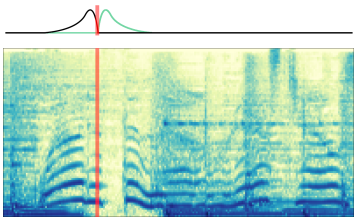
dimensional data like spectrograms, it does not seem a good idea to use MLPs since the resulting model can easily become huge and one might inquire the risk of overfitting. A common solution to this issue consists in using CNNs when processing high-dimensional data like spectrograms.



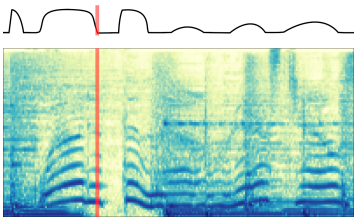
(a) All spectrogram bins are used to compute **MLP** outputs.



(b) The black area represents the receptive field of a **CNN** filter.



(c) The **RNN**s'capacity to model long-term dependencies is limited. Its capacity decreases with time.



(d) **Attention**-based models can encode long-term dependencies from the future and the past.

Figure 2.2: Spectrograms (vertical axis: frequency – horizontal axis: time) to graphically illustrate the merits of each artificial neural network architecture. The vertical red line in (c) and (d) denotes an arbitrary time-step t where we want to utilize additional temporal information. Black and green lines illustrate how reachable is each time-step: the higher the line, the higher the amount of information that can be employed by the model at time t. The black line in (c) shows the attention weights' $\alpha_{(t,n)}$ values at time t.

Figure 2.2b depicts the receptive field of a vertical CNN filter capable to encode timbral traces. The black area (the receptive field) can be interpreted as the number of parameters to be learned by this model. Hence, it is not difficult to understand that the size of the model is significantly reduced when using CNNs (only the black area *vs.* all the spectrogram bins). A direct consequence of having a smaller model is that it might have more chances to generalize. CNN models are smaller because the convolution operation shifts CNN filters horizontally and vertically throughout the spectrogram. Consequently, for spectrogram-based CNNs, the filter weights are shared across time and frequency. The arrows in Figure 2.2b denote the horizontal and vertical shifts performed by the CNN. Since spectrogram-based CNNs convolve across time (horizontal shift) and frequency (vertical shift), these can capture representations that are time and frequency invariant by construction (Pons et al., 2017b).

Figure 2.2c graphically illustrates how RNNs are not able to learn long-term dependencies due to the "vanishing/exploding gradient" problem. We highlight this phenomena with the **black** & **green** lines above the spectrogram, where we showcase the amount of **past** & **future** information that could be employed by a RNN at time t (denoted by the red vertical line). Note that one can feed future information into RNNs by simply changing the temporal direction we look at: for example, by changing $\mathbf{h}^{(t-1)} \rightarrow \mathbf{h}^{(t+1)}$ in Eq. 2.12. In contrast, Figure 2.2d shows that for a given time t an attention-based model can employ information from any time position along the spectrogram. Note the black line above the spectrogram, where we show the attention weights' $\alpha_{(t,n)}$ values. The weighted average mechanism (through time considering the $\alpha_{(t,n)}$ values) employed by the attention models

enables a *direct path* where long-term dependencies can flow. Finally, we want to remark that the shapes of the black and green lines in figures 2.2c and 2.2d were invented just for didactic purposes.

Although in previous paragraphs we assumed single-layered models processing spectrograms and discussed each architecture separately, these are normally stacked one on top of the other to construct deep models. For example, a common pipeline consists of *(i)* stacking several CNN layers to extract local features from spectrograms, *(ii)* RNNs or attention are used to aggregate the CNN feature maps across time, and *(iii)* the temporally aggregated signal is used to predict what's intended with a MLP. Note that this deeper pipeline allows to learn musical characteristics at different time-scales, and to take advantage of the merits of each architecture (Raffel, 2016; Choi et al., 2017a).

2.2 History of artificial neural networks

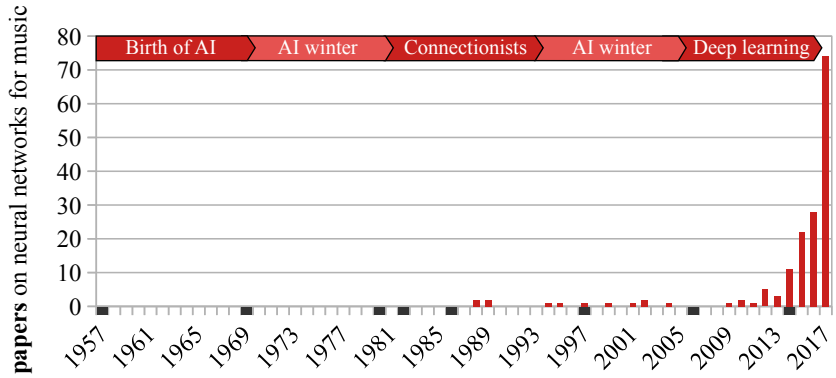
Neural networks are a machine learning technique that is often associated with artificial intelligence (AI) due to its capacity to learn in a data-driven fashion, without heavily relying on human domain expertise. Possibly due to that, many people erroneously interchange these two terms. However, it is undeniable the contribution of neural networks to the field of AI. For that reason, and to provide some context to the reader, before diving into the history of neural networks for music (in Section 2.3), we first review which are the main historical contributions of neural networks to the field of AI.

2.2.1 Birth of AI & the first AI winter: the 60's and 70's

The term artificial intelligence (AI) was proposed at a workshop held in Dartmouth College in 1956 to discuss computers, natural language processing, neural networks, theory of computation, abstraction and creativity (McCarthy et al., 1955). Provided that it is where the field coined its name, this workshop is considered the birthplace of AI. Important names assisted to this rather small (11 attendees) influential meeting: Minsky, Mackay, Shannon, Newell, or Simon, among others. Starting from this historical milestone, we will go through the most influential articles on artificial neural networks till date (see the black-listed items in Figure 2.3 for a graphical depiction of the chronology we will cover).

The early days of AI were an era of discovery and optimism, what motivated private and public agencies to raise funding for research. One of the first highly influential inventions in the field was the perceptron. It was proposed by Rosenblatt (1957), and it is the basis of modern artificial neural networks. However, after several years of research, their outcomes were far from the original expectations. An illustrating example can be the work by Minsky and Papert (1969), who found that a simple XOR function cannot be implemented by a perceptron (a state-of-the-art model back in the 60's).[6] Provided that their actual technology was behind everyone's expectations, the research funding started to decrease delving AI researchers into their first winter.

[6] One actually needs a more expressive model like a MLP to implement a XOR function (that is not linearly separable).



1957 Invention of the perceptron
1969 Discovery: XOR functions cannot be implemented by a perceptron
1980 Invention of the CNN
1982 Invention of the RNN
1986 The backpropagation algorithm was proposed to train MLPs
1988 MLP and RNN applied to algorithmic music composition
1995 MLP applied to music audio classification from spectrograms
1997 Invention of the LSTM
2002 LSTM applied to algorithmic music composition
2006 Discovery: how to systematically train deep neural networks
2009 CNN applied to music audio classification from spectrograms
2014 End-to-end learning for music audio classification
2014 Invention of attention-based artificial neural networks
2015 Attention-based model applied to audio-to-midi music matching

Figure 2.3: Chronology of the most influential articles in neural networks (in **black**) for music (in **red**).

2.2.2 Connectionists & the second AI winter: the 80's and 90's

It was not until the 80's that AI strikes back. The main success of this generation was to use "expert systems" to solve task-specific problems with symbolic AI. These systems were able to solve important practical challenges via restricting themselves to a small domain of specific knowledge where is possible to craft the knowledge-base required for most symbolic AI systems (Feigenbaum, 1981).

Besides the actual success of "expert systems", novel ideas around neural networks were introduced at that time. During this decade, CNNs (Fukushima and Miyake, 1980) and RNNs (Hopfield, 1982) were invented. And, importantly, the backpropagation algorithm was proposed to train MLPs (Rumelhart et al., 1986). This last contribution was particularly relevant because via enabling the training of multi-layered (deep) neural networks, the field could get over the limitations of the perceptron and, e.g., artificial neural networks can be trained to learn the XOR function. These people working on neural networks during the 80's are known as connectionists, a term motivated by the nature of neural networks (where several simple units are connected to build a powerful model). However, the computing power required to successfully train these models was not yet available, what severely limited the possibilities of connectionists. For that reason, during the following years, connectionists entered in a second AI winter.

During the 90's, other competing methods have dominated the field of machine learning. A prominent alternative were, and still are, the support vector machines (SVMs) by Cortes and Vapnik (1995). These are powerful classifiers that tend to perform competently while being "small" (with few trainable parameters). In contrast, artificial neural networks tend to be significantly "larger", which makes them hardware demanding and prone to overfitting. Hence, the resources required to train SVMs are much less than for artificial neural networks, what implied a practical advantage during the 90's. However, SVMs assume that the input data is linearly separable (Cortes and Vapnik, 1995). In order to fulfill this requirement as much as possible, the SVMs' input is generally formed by a set of carefully selected fea-

tures. Differently from SVM classifiers, artificial neural networks can jointly learn the feature extractor and the classifier, which reduces the amount of human domain expertise required to build SVM classifiers (relying on engineered features). Unfortunately, though, the computing power required for successfully training "large" artificial neural network with the capacity to learn a feature extractor + classifier was not yet available during that decade, what favored the success of SVMs.

2.2.3 Deep learning: the early 2000's

In short, deep learning models are multi-layered artificial neural networks. However, training these can be intricate. Although the backpropagation algorithm (employed to train deep neural networks) was proposed during the 80's (Rumelhart et al., 1986), there were some practical issues that made deep neural networks difficult to train. For backpropagation to work, it was required to carefully initialize $\boldsymbol{\theta}$ and to sensibly search along several hyper-parameters (Glorot and Bengio, 2010). Hence, training deep neural networks was prone to failure because there was no clear understanding of how on systematically train those.

As seen, two main obstacles were limiting the wide adoption of deep artificial neural networks: *(i)* they were difficult to train, and *(ii)* not much computing power was available at that time. An encouraging solution for the first challenge came by Hinton et al. (2006), who proposed a method that allows to systematically train deep artificial neural networks. Their method is based on a layer-wise unsupervised pretraining schema that enables to properly initialize deep artificial neural networks. As a result of having a reasonable $\boldsymbol{\theta}$ initial-

ization, deep neural networks are now easier to train and can work in practice. The solution to the second challenge came in the form of parallel computing with GPUs. Krizhevsky et al. (2012) won a prestigious image classification competition by a large margin using deep CNNs. One of their keys to success was to develop a GPU implementation that enabled them to scale up the problem of training deep artificial neural networks. These two seminal papers (Hinton et al., 2006; Krizhevsky et al., 2012) are considered the start of the deep learning era, that heavily impacted many fields and stimulated very creative work in the area of neural networks. For example: attention-based artificial neural networks were recently proposed in the context of machine translation (Bahdanau et al., 2014), or modern generative models are now capable to produce appealing images and structured text (Brock et al., 2019; Radford et al., 2019).

2.3 History of artificial neural networks for music

After reviewing the main historical milestones in artificial neural networks research, now is time to dive into the history of neural networks for music. The red vertical bars in figures 2.3 and 2.4 depict the number of papers on artificial neural networks for music that were published throughout the years.[7] On it, we can clearly identify two main regions: the connectionists era (1988-2009), and the deep learning era (since 2009). In the following, we will outline their main contributions.

[7] https://github.com/ybayle/awesome-deep-learning-music

2.3.1 Connectionists era: from 1988 to 2009

During the second AI winter, a series of spurious work maintained the field's relevancy from 1988 to 2009. This is the contribution of the so-called connectionists to the field of artificial neural networks for music, a work that is pretty much unknown to most contemporary researchers.

The first wave of work was initiated in 1988 by J. P. Lewis and Peter M. Todd, who proposed the use of artificial neural networks for automatic music composition. On the one hand, Lewis (1988) used a MLP for his algorithmic approach to composition called "creation by refinement". That is, in essence, based on the same idea as DeepDream[8]: utilizing gradients to create art. On the other hand, Todd (1988) experimented with auto-regressive RNNs to generate music sequentially. Many people kept using this same auto-regressive principle throughout the years. Among them: Eck and Schmidhuber (2002), who proposed using LSTMs for sequential algorithmic composition; or, to consider a more recent work, the Wavenet model (van den Oord et al., 2016) —which is capable of generating *music* directly in the waveform domain— also makes use of this same sequential (causal) principle. Hence, the old connectionist ideas that Todd and Lewis introduced back in the 80's for algorithmic composition are still valid today. But, if their principles were correct, why did they not succeed? In Lewis' words: "it was difficult to compute much of anything". While a modern GPU may have more than 100 tflops of theoretical performance, a VAX-11/780 (the workstation he used back in 1988 for his work) had 0.1 mflops.

[8] https://github.com/google/deepdream

But let's go back to discuss the work of Eck and Schmidhuber (2002). In their paper "Finding temporal structure in music: blues improvisation with LSTM", they try to address one of the major issues that algorithmically composed music had (and still has): the lack of global coherence or structure. To address this challenge, they proposed the use of LSTMs — which are supposedly better than vanilla RNNs at learning long temporal dependencies. Note, then, that as a result of this experimentation, music has been one of the early applications of LSTMs (Hochreiter and Schmidhuber, 1997). However, interestingly, Eck and his collaborators have recently moved from using LSTM-based models to attention-based ones for algorithmic composition (Huang et al., 2019). They found that their attention-based model can learn "long-term structure" that "captures global timing, giving rise to regular phrases", what confirms how powerful are attention-based models for learning long-term dependencies in music (see Section 2.1.7 for further details on this discussion).

Although most connectionists were working on algorithmic music composition, some researchers also explored other music tasks. Laden and Keefe (1989) did some work on chord classification with MLPs, where they were classifying chords (e.g., the C major chord: C-E-G) into chord types (major, minor, and diminished). And Dannenberg et al. (1997) studied how to use MLPs to classify MIDI scores into music styles like "syncopated" or "pointillistic". As a consequence, automatic music composition, chord recognition and music classification are the first musical tasks that were ever addressed with artificial neural networks.

As seen, an important trend among connectionists was to work with symbolic music data. However, there were some remarkable

exceptions. Matityaho and Furst (1995) explored for the very first time to work with spectrograms. Their goal was to distinguish between pop and classical music with MLPs. And Marolt et al. (2002) used spectrograms for the task of note onset detection with MLPs. These early works were operating on top of spectrograms, a relatively low level signal, as a way to jointly learn the feature extractor and the classifier with MLPs. After those publications (Matityaho and Furst, 1995; Marolt et al., 2002) a new research period started: a race had begun to be the first to address any task in an end-to-end learning fashion — what means learning a mapping system (or learnable function) able to solve a task directly from a low level representation of the signal, ideally from the waveform. Solving a task directly in the raw audio (waveform) domain would allow removing any pre-processing step, enabling to minimize the amount of human intervention and the assumptions done with respect to the input signal. Note that the more computing power and data are available for training, the more effective these models are at processing lower level signals (like waveforms or spectrograms). In Figure 2.4 we graphically depict this trend: since 1988 it was common to utilize symbolic music data (Todd, 1988; Lewis, 1988); in 1995 researchers started using lower level signal representations (like spectrograms) as input to their models (Matityaho and Furst, 1995); and it was not until 2014 that someone successfully trained an end-to-end model employing raw audio data (Dieleman and Schrauwen, 2014).

That said, connectionists were not only interested in using artificial neural networks for end-to-end learning. Another (very popular) line of research was to utilize artificial neural networks as classifiers (Kaminsky and Materka, 1995; Kostek and Krolikowski, 2014;

Herrera-Boyer et al., 2003). One important focus of these works was to carefully select useful features for the task at hand. Provided that these engineered features are discriminative for a given task, artificial neural networks can be employed as classifiers. Consequently, the resulting models can be smaller and more tractable. As a result, many explored the use of artificial neural networks as classifiers throughout the years. The early works on classification were mostly on instrument recognition: Kaminsky and Materka (1995) build a classifier that its main feature was the energy envelope; and Kostek and Krolikowski (2014) used a richer set of features containing spectral and temporal descriptors (e.g., brightness, energy of the harmonics, or attack-time). Besides instrument classification, the connectionists also addressed other tasks with artificial neural network classifiers. For example, Soltau et al. (1998) build a genre classifier based on cepstral features.

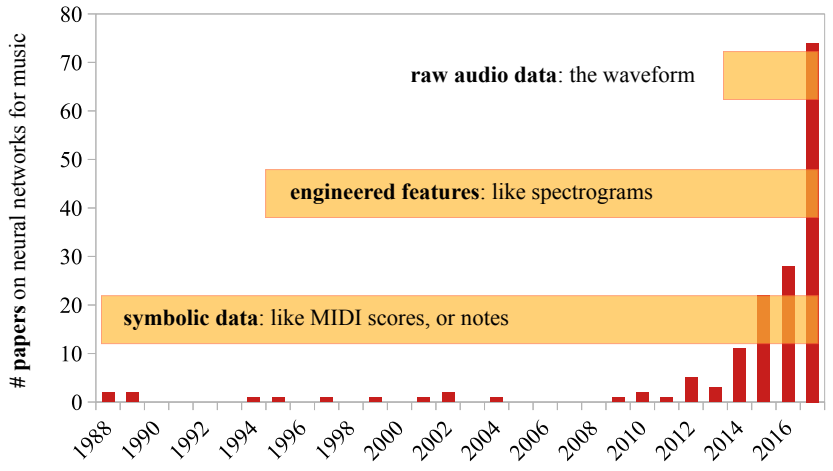


Figure 2.4: Data format trends in artificial neural networks for music. The more computing power and data are available for training, the more effective these models are at processing lower level signals.

2.3.2 Deep learning era: since 2009

Around 2009, the second AI winter ended for music technologists and the first bunch of deep learning works started to impact the field. As one can observe in Figure 2.3, after the deep learning era began, the field of artificial neural networks for music has grown exponentially. During that time, researchers started to successfully tackle more complex problems —like music audio tagging (Lee et al., 2009; Dieleman and Schrauwen, 2014), chord recognition (Humphrey and Bello, 2012), or source separation (Chandna et al., 2017; Lluís et al., 2018)— with deep artificial neural networks.

One first influential work of the deep learning era was by Lee et al. (2009), who build the first deep spectrogram-based CNN for music genre classification. This is the foundational work that established the basis for a generation of deep learning researchers who spent great efforts designing CNN models to better recognize high-level concepts from music spectrograms.

However, not everyone was satisfied utilizing spectrogram-based models. For example, Dieleman and Schrauwen (2014) explored the ambitious research direction of end-to-end learning in music audio. They were the first to investigate waveform-based models for the task of music audio tagging. Although they demonstrated some degree of success, spectrogram-based models were still superior to waveform-based ones. At that time, not only were the models not mature enough, but also the training data was scarce when compared to the amounts of data now some companies have access to. For example, a recent study run at Pandora Radio shows that waveform-based models can outperform spectrogram-based ones provided that enough training data are available (Pons et al., 2018).

Another historically remarkable piece of work comes from Humphrey et al. (2013), who convinced LeCun to co-author the deep learning for music *manifesto*. In their article, the authors convey the power of learning deep representations from data. Interestingly, they argue that the community was already employing deep features. For example, the widely used MFCCs features (based on the discrete cosine transform of the log-mel power spectrogram) are already a deep representation, in the sense that several layers of transformations are required to compute those features (Davis and Mermelstein, 1980).

Broadly speaking, we have structured the connectionists' work into two main application areas: *(i) music information retrieval*, which aims to design models capable to recognize the semantics present in music signals; and *(ii) algorithmic music composition*, with the goal to computationally generate new appealing music pieces. These two application areas are still very active, and have been a subject of research during the deep learning era. Recent deep learning works in *(i) music information retrieval* are still pushing the boundaries of what's possible via improving the architectures that define these models (Pons et al., 2016b; Lee et al., 2018). But actual researchers do not only intend to improve the performance of such models. They are also studying how to increase their interpretability, or how to reduce their computational footprint (Pons et al., 2017b). Furthermore, as previously mentioned, there is a strong interest in designing architectures capable to directly deal with waveforms for a large variety of tasks. However, researchers have not yet succeeded in designing a generic strategy that enables waveform-based models to solve a wide range of problems — something that would allow, for example, the broad applicability of end-to-end audio classifiers (Lee

et al., 2018). The area of *(ii) algorithmic music composition* has also been very active during the deep learning era. Remember that back in the 80's (Lewis, 1988; Todd, 1988) and during the early 2000's (Eck and Schmidhuber, 2002), rather simplistic auto-regressive neural networks were used. But now is the time for attention-based artificial neural networks (Huang et al., 2019), and modern generative models like generative adversarial networks (Goodfellow et al., 2014a) or variational auto-encoders (Kingma and Welling, 2013). Interestingly enough, these modern generative models are not only being used to compose novel scores in symbolic format, but models like waveform-based generative adversarial networks (Donahue et al., 2019; Engel et al., 2019) or Wavenet (van den Oord et al., 2016; Engel et al., 2017) are also being used to explore novel timbral spaces or to render new songs directly in the waveform domain (as opposed to composing novel MIDI scores).

As seen, most connectionists' works were processing symbolic music data. During the connectionists era only baby steps were done in the direction of end-to-end learning, and just a few works (Matityaho and Furst, 1995; Marolt et al., 2002) explored to directly process spectrograms with artificial neural networks — most possibly due to hardware and data limitations. However, during the deep learning era, end-to-end learning models are becoming more and more popular. For example, end-to-end learning is enabling novel ways to approach the music source separation problem (Cano et al., 2019). The music source separation task consists in estimating the original music sources (multi-track recordings) from a mixture, and a common approach consists in filtering the mixture signal with time-frequency masks that are predicted by an artificial neural network (Jansson

et al., 2017; Chandna et al., 2017). Consequently, most source separation techniques employ spectrogram-based models, and are therefore (by default) omitting part of the signal: the phase. Interestingly, end-to-end learning allows to address these historical challenges via bypassing the problem. By operating directly on the waveform domain, the model can take into account all the information available in the raw audio signal (including the phase), and also allows to move beyond the current filtering paradigm (Stoller et al., 2018; Lluís et al., 2018). But we can also observe this trend in other tasks of relevance like in automatic music transcription. This task is particularly challenging because it comprises several subtasks; including multipitch estimation, onset and offset detection, instrument recognition, beat and rhythm tracking, interpretation of expressive timing and dynamics, and score typesetting (Benetos et al., 2019). A traditional approach to automatic music transcription would consist in designing specialized subsystems for each of the subtasks, to later put all the pieces together. Instead, end-to-end learning brings the opportunity to directly map music audio into music notation without explicitly modeling most of the subtasks described above (Vogl et al., 2017; Southall et al., 2017; Hawthorne et al., 2018).

Unfortunately, a limitation of end-to-end systems is that these require large amounts of training data to perform reasonably (Pons et al., 2018; Stöter et al., 2018) — what limits the applicability of this approach in many scenarios. For that reason, many successful deep learning models are built on top of engineered features (like spectrograms) (Choi et al., 2018b). By using a higher-level input representation of the signal, the available training data is not employed to learn low-level features but to directly learn how to address a task from an

already informative feature. In other words, the input restricts the solution space in a way that data can be used more efficiently to *just* solve the task (e.g., to classify music genres from already discriminative audio features). Another direction to address the data scarcity issue consists of tailoring these models towards learning solutions closer to what humans perceive. For example, by designing musically or perceptually inspired deep learning architectures (Pons et al., 2016b; Ravanelli and Bengio, 2018). These approaches aim to place the model close to a plausible solution (having the generalization power of known priors) to employ the training data more efficiently. As seen, until large corpus of annotated audio are not available for research, many are exploring to restrict the solution space of deep learning models so that these have more chances to generalize with unseen data. Additionally, researchers are exploring the use of data augmentation and unsupervised learning for accessing to virtually infinite amounts of training data. While data augmentation techniques have already been successfully used by many (Schlüter and Grill, 2015; McFee et al., 2015; Miron et al., 2017), it does not exist a consensus on how to efficiently leverage unlabeled data with unsupervised learning (Lee et al., 2009; Slaney et al., 2008; Freitag et al., 2017).

2.4 Summary and conclusions

Before diving into the history of neural networks for music, we have introduced and discussed the main artificial neural network architectures from the perspective of a music technology practitioner. We observed that *deep* neural networks are a suitable tool for modeling

music computationally. Music is hierarchic in frequency (note, chord, etc.) and in time (onset, rhythm, etc.), and deep neural networks naturally allow for this hierarchic representation since their architecture is hierarchical due to its depth. Furthermore, relationships between events in the time domain are important for modeling music. To this end, we have seen how attention-based models can capture long-term dependencies (*e.g.*, music structure or music motives) while CNNs and/or RNNs can model the local context (*e.g.*, timbral traces, onsets or rhythm patterns). However, besides the above conceptual motivation, during these last years a vast amount of empirical results are also showing the potential of using deep neural networks for music (Huang et al., 2019; Pons et al., 2018; Stöter et al., 2018; van den Oord et al., 2016; Stoller et al., 2019). As a result, the core techniques used in the field of music informatics have moved from feature engineering to deep learning (Humphrey et al., 2013). In concordance with that, the number of publications using artificial neural networks for music has grown exponentially (see the red bars in Figure 2.3).

Later, we reviewed the history of artificial neural networks for music — that can be divided into two periods: the connectionists era (1988-2009), and the deep learning era (since 2009). Besides the technical innovations that enabled scientific progress in the field, we noted that artificial neural networks require a significant amount of data and computing power to be competitive. While back in the connectionists era the computing constraints were severely limiting the outcome of artificial neural networks research, hardware restrictions are no longer limiting the deep learning generation. However, strong resource asymmetries exist between the industry and the academy. While industry labs seem to build on top of abundant resources, aca-

demic institutions are slowly building the required infrastructure for deep learning. An important asset for this public infrastructure are the datasets. Building public datasets for easy benchmarking is of capital importance to enable and to measure progress in the field. Despite the recent initiatives on building public datasets for general audio tagging (Fonseca et al., 2017; Gemmeke et al., 2017) or music source separation (Rafii et al., 2017), it still exists the need of building large datasets for a variate set of tasks (Cano et al., 2019; Benetos et al., 2019; Mueller et al., 2019).

We also described an "end-to-end learning trend" among deep learning researchers, who are currently exploring the possibilities of this idea. If possible, end-to-end learning enables interesting mappings from *waveform-to-classes*, or even from *waveform-to-waveform*. Hence, it is time to re-define what's possible and what's not. Artificial neural networks are now enabling tools and novel approaches that were previously unattainable, and important tasks like music audio tagging (Pons et al., 2018; Lee et al., 2018), music source separation (Lluís et al., 2018; Stoller et al., 2018; Pons et al., 2016a), automatic music transcription (Vogl et al., 2017; Southall et al., 2017), or algorithmic music composition (Dieleman et al., 2018) are now being revisited from the end-to-end, deep learning, perspective.

Current efforts are also centered in exploring innovative ways to put the pieces together. For example, artists are using artificial neural networks as an instrument for creativity: to create novel timbral spaces (Engel et al., 2017), or to assist the music creation process with automatically rendered compositions (Huang et al., 2019). Besides, innovative uses of artificial neural networks can also lead to new ways for humans to interact with music (Goto and Dannenberg, 2019). For

example, deep learning greatly facilitates working with multi-modal data. In part, because these powerful data-driven models can learn the relations between the different music modalities in an end-to-end fashion. Following this direction could, e.g., enable enriched musical experiences where audio excerpts, videos, and editorial metadata could be jointly represented in a shared cross-modal space where a user could query additional information like upcoming concerts, videos, lyrics, or the biography of an artist (Mueller et al., 2019).

2.5 Contributions

Throughout our research, we identified two important literature gaps that aim to be covered along with this introductory chapter: *(i)* the lack of a historical review covering the field of artificial neural networks for music; and *(ii)* the lack of a music-specific approximation to the field of artificial neural networks.

We consider that an introductory journey through the history of artificial neural networks for music can be useful for beginners to understand which are the historical challenges, the current state-of-the-art and its limitations, and possible future directions. For this reason, we also prepared a blog-post[9] to further disseminate the above introduced history of artificial neural networks for music.

[9] www.jordipons.me/neural-networks-for-music-history/

Chapter 3

Musically Motivated CNNs for music tagging

A common criticism of deep learning relates to the difficulty in understanding the underlying relationships that neural networks are learning. Consequently, these are generally considered "black-boxes" that are challenging to interpret. In this chapter, we explore various architectural choices of relevance for music signals in order to gain intuition towards understanding what the chosen architectures are learning. To motivate the design strategy we propose, we first discuss how CNN filters with different shapes can fit specific musical concepts (see Section 3.1) and, based on that, we propose several musically motivated architectures. Despite the efforts on trying to puzzle out what the networks are learning, it is still not clear how to navigate through the network parameters space. It is hard to discover the adequate combination of hyper-parameters for a particular musical task, which leads to architectures being difficult to interpret. Given this, our work aims to rationalize the design process by

proposing musically motivated architectures. To this end, we study how CNNs can be tailored towards learning generalizable musical concepts, as a way to study which deep learning architectures can be most appropriate for modeling music computationally.

As a proof of concept, in Section 3.2 we assess the proposed design strategy. For this first experiment, we evaluate several CNNs (designed to learn temporal and timbral representations) against a dataset that is known for having classes that are well represented by their tempo. Within that context, e.g., we can investigate weather "temporal" CNNs achieve better results than "timbral" CNNs. With this preliminary experiment, we note the potential of this approach that we investigate further in Sections 3.3 and 3.4 — where we study how to efficiently learn temporal & timbral representations with CNNs.

3.1 Introductory discussion: CNN filter shapes are important

As seen in Chapter 2, several architectures can be combined to construct a deep learning model. However, throughout this chapter we focus on CNNs. Provided that our goal consists of tailoring deep learning models towards learning musically relevant features, CNNs seemed an intuitive choice given that most musically relevant features are local (e.g., timbre, onsets, rhythm, or tempo). Furthermore, CNNs fed with spectrograms allow to design CNNs filters having interpretable dimensions in the first layer: time and frequency, what allows designing musically motivated architectures. For these reasons, and provided that spectrogram-based CNNs are widely used among deep learning researchers (Choi et al., 2016; Schlüter and Böck, 2014), we frame our discussion within that context.

Regardless of the competitive results achieved and the conceptual benefits of using deep learning approaches (see Chapter 2), we still do not fully grasp what neural networks are learning. Back in 2016, when we proposed this line of research, only few works tried to give an explanation of what deep learning models were learning. For example, Sander Dieleman[1] did some work showing that higher-level concepts can be identified from lower-level music representations. He found that the first CNN layers of their music recommendation system had filters specialized in low-level musical concepts (like vibrato, vocal thirds, pitches, chords), whereas upper CNN layers had filters specialized in higher-level musical concepts (like christian rock, chinese pop, 8-bit). This matches with similar results found by the image processing research community where lower layers are capable of learning shapes that are combined in higher layers to represent objects (Zeiler and Fergus, 2014). Furthermore, Dieleman et al. (2011) also proposed a deep learning algorithm that preserves musically significant timescales (beats-bars-themes) within the architecture itself, what "leads to an increase in accuracy" for music classification tasks and gives an intuition of what the network may be learning, showing that musically motivated architectures may be beneficial for modeling music computationally. Moreover, Choi et al. (2015) proposed a method for *auralisation* —which is an extension of the CNNs visualization method Zeiler and Fergus (2014)— that allows to interpret by listening what each CNN filter has learned.

Due to the CNNs success in computer vision, its literature significantly influenced music technologists. For image processing, small squared filters of 3×3 or 12×12 are widely used (He et al., 2016). As

[1] http://benanne.github.io/2014/08/05/spotify-cnns.html

a result of that, music technology researchers tend to use similar filter shape setups (Choi et al., 2016; Schlüter and Böck, 2014). However, note that the image processing filter dimensions have spatial meaning, while spectrogram-based filter dimensions correspond to time and frequency. Therefore, *wider* filters may be capable of learning longer temporal audio dependencies while *higher* filters may be capable of learning more spread timbral features. In order to motivate researchers to be conscious about the potential impact of choosing one filter shape or another, three examples and a use case are discussed in the following. Throughout this chapter we assume the spectrogram dimensions to be $M \times N$ and the filter dimensions to be $m \times n$. We center our discussion around the first CNN layer, where the filter dimensions are interpretable. Hence, M and m stand for the number of frequency bins and N and n for the number of time frames.

- Squared/rectangular filters ($m \times n$ filters) are capable of learning time and frequency features at the same time, and are widely used by music technology researchers (Choi et al., 2016; Schlüter and Böck, 2014). Such filters can learn different musical aspects depending on how m and n are set. For example, the bass could be well modeled with a small filter ($m \ll M$ and $n \ll N$, representing a sub-band for a short-time) because this instrument is sufficiently characterized by the lower bands of the spectrum, and the temporal evolution of the bass notes is not so long. An interesting interpretation of such small filters is that they can be considered pitch invariant to some extent. Since the convolution also happens in the frequency domain, such filters can be modulated to represent different pitches. However, note that such pitch invariability would not

hold for instruments having a large pitch range since the timbre of an instrument changes accordingly to its pitch. As another example, let's consider the cymbals or the snare drums sound sources. These could be suitably modeled by setting $m = M$ and $n \ll N$, because they are broad in frequency and have a fixed decay time. Interestingly, a bass could also be modeled with this filter — however, the pitch invariance interpretation would not hold because its dimensions ($m = M$) would not allow the filter to convolve along the frequency axis. Consequently, pitch will be encoded together with timbre.

- Temporal filters ($1 \times n$). By setting the frequency dimension to m=1, such filters won't be capable of learning frequency features — but will be specialized on finding relevant temporal cues to address the task at hand. However, although the filters themselves are not learning frequency features, deeper layers may be capable of exploiting frequency cues (since the frequency interpretation still holds for the resulting activations because the convolution operation is done bin-wise with m=1). Musically speaking, one expects those temporal filters to learn rhythmic/tempo patterns within the analyzed frequency band.

- Frequency filters ($m \times 1$). By setting the time dimension to n=1, such filters won't be capable of learning temporal features — but will be specialized in modeling the frequency features that are relevant for the task at hand. Similarly as for the temporal filters, upper layers could still find some temporal dependencies from the resulting activations. From the musical perspective, one expects these frequency filters to learn pitch, timbre or equalization setups, for example.

Moreover, we would like to discuss Choi et al. (2015) results' as a use case.[2] They trained a 4-layer CNN of squared 12×12 filters to address the task of music genre recognition. After auralising and visualizing the learnt filters', they conclude that their deep learning model was capable of: finding attack/onsets, selecting bass notes and separating kick drums. As previously discussed, squared small filters may be capable of modeling instruments appearing in a sub-band (e.g., bass and kick), and also to model temporal features (e.g., onsets) due to its length. Also, it is not surprising that they did not found the trained CNN to model, e.g., cymbals or snare drums — what may be definitely challenging to be modeled by a CNN with such small filters.

3.2 Proof of concept: the ballroom dataset

After noting the potential impact of choosing one CNN filter shape or another, in this section we explore how these ideas can be translated into an actual design strategy that allows using our domain expertise on music-spectrograms to inform our choices when designing CNNs.

Our goal is to guide spectrogram-based CNNs towards learning musically relevant representations by carefully designing the filter shapes in the first layer. In that way, the proposed CNNs can efficiently capture the local context that is relevant for the task at hand. As a direct consequence of approaching the CNNs' design in that way, we will see that *(i)* the learnt CNN filters are more interpretable, and *(ii)* the number of learnable parameters can be greatly reduced (what might increase the generalization capabilities of the learnt models in scenarios where few training data are available).

[2] http://keunwoochoi.blogspot.com.es/2015/10/ismir-2015-lbd.html

Motivation: audio material

This proof of concept is realized using the Ballroom dataset.[3] It consists of 698 tracks, of around 30 seconds long, divided into 8 music genres: cha-cha-cha, jive, quickstep, rumba, samba, tango, viennese-waltz and slow-waltz (Gouyon et al., 2004). This dataset is particularly interesting *(i)* due to its small size, and *(ii)* because its classes are highly correlated to tempo. This allows studying the impact of employing musical domain knowledge (e.g., knowing that temporal cues are important) in these scenarios where no large datasets are accessible for training. Even though this dataset was originally designed for rhythmic patterns classification, Gouyon et al. (2004) already showed in the original dataset publication that Ballroom classes are rather well characterized by their tempo. Note that a k-nearest neighbor on top of BPM annotations achieves 82.3% accuracy. Hence, tempo and rhythm are relevant cues when predicting Ballroom classes. Inspired by this data characteristic, our proof of concept consists in evaluating several musically motivated CNNs designed to capture temporal and timbral features — to assess weather, as one might expect, the former outperforms the latter.

Besides, this dataset has been extensively used. For example, Marchand and Peeters (2016) achieved 96% accuracy — and we set this algorithm as baseline for our deep learning methods using time and frequency cues. However, not all our models employ time and frequency cues as Marchand and Peeters (2016). One of our models (*Time*) is designed to focus on temporal cues. In this case, to allow a fair comparison, we set the k-nearest neighbor by Gouyon et al. (2004) as baseline: 82.3% accuracy. Besides, we also propose another

[3] http://mtg.upf.edu/ismir2004/contest/tempoContest/node5.html

architecture (*Frequency*) that is designed *not* to focus on temporal cues. Provided that temporal cues are of importance to predict Ballroom classes, we do not expect this model to perform competently. In this case, we set a random baseline based on the probability of guessing the most likely class: cha-cha-cha (15.9%).

Experimental setup

The audio is fed to the model through fixed-length log-mel spectrograms that are N frames wide. Interestingly, Dieleman and Schrauwen (2014) used raw audio waveforms as input to their deep neural network and found that the lowest CNN layer was learning "frequency-selective features covering the lower half of the spectrum" — what motivates our choice, since mel filter-banks allocate more frequency resolution to the lower parts of the spectrum. Besides, log-mel spectrograms are widely used as input to neural network classifiers because these tend to perform competently (Dieleman et al., 2011; Choi et al., 2015). Throughout this work we use 40 mel bands derived from a STFT-spectrogram computed with a Blackman Harris window of 2048 samples (50% overlap) at 44.1 kHz. A dynamic range compression is applied to the input mel spectrograms in the form of $log(1 + C \cdot x)$, where $C = 10.000$ is a constant controlling the amount of compression (Dieleman and Schrauwen, 2014). The resulting spectrograms are normalized so that the whole dataset spectrograms (together) have zero mean and variance one. Note that this normalization is not attribute-wise, as this would perturb the relative information encoded between spectrogram bins/frames. Three architectures are proposed to experiment with musically motivated CNNs:

(i) ***Black-box*** **architecture**: is based on previous work using

CNNs for the task of music/speech classification (Lidy, 2015). In this setup (that obtained the best results of the MIREX 2015 task on music/speech classification[4]), a single CNN layer with fifteen 12×8 filters and a 2×1 max-pool layer had been used, followed by a feed-forward layer of 200 units connected to an output softmax layer. We adapted this approach to further get better results by slightly changing the previous setup to thirty-two 12×8 filters and a max-pool layer of 4×1.

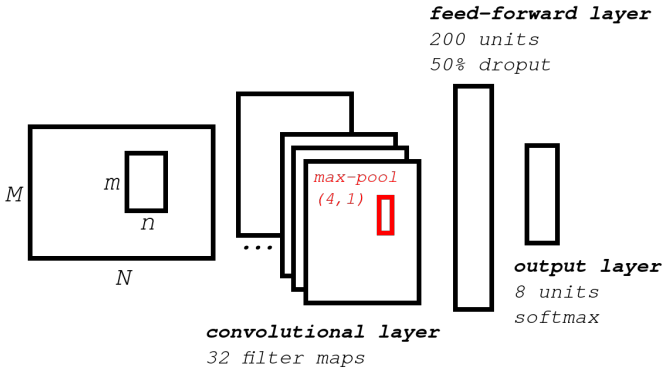


Figure 3.1: The *Black-box* architecture.

(ii) ***Time*** **architecture**: is designed to learn temporal representations. It is composed of a CNN layer of 32 temporal filters $(1 \times n)$ followed by a max-pool layer of $(M \times 1)$ connected to an output layer. The fact that the max-pool layer spans all over the frequency axis (m=M) and covers only one frame (n=1), allows to only encode temporal content (due to the summarization done along the frequency axis) and to preserve the frame resolution, respectively.

[4]http://www.music-ir.org/mirex/wiki/2015:Music/
/Speech_Classification_and_Detection_Results

(iii) ***Frequency* architecture**: is designed to learn frequency representations. It is composed of a CNN layer of 32 frequency filters ($m \times 1$) followed by a max-pool layer of ($1 \times N$) connected to an output layer. The max-pool layer (with n=N) operates similarly as in the *Time* architecture, but in that case the summarization is in time. Note that the extreme case of a *Frequency* architecture would be to input only one frame. However, we expect the statistics provided by the max-pool layer to help the network learning better timbral representations.

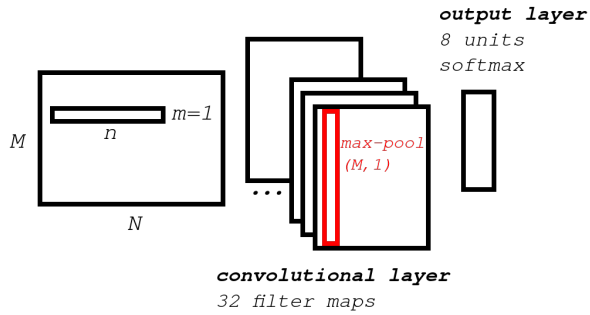


Figure 3.2: The *Time* architecture.

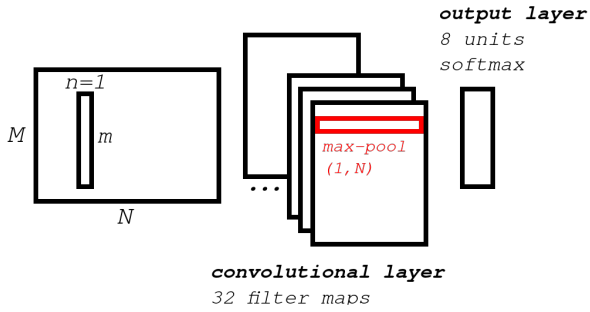


Figure 3.3: The *Frequency* architecture.

The *Black-box* architecture follows a standard architecture that one could find in the literature back in 2016 (Choi et al., 2015; Schlüter and Böck, 2014). We call it *Black-box* because there is no musically-relevant reason for such architectural choices. Note that the learnt CNN filters are not straight-forward to interpret because there is not apparent motivation behind their design. *Time* and *Frequency* are introduced as musically inspired systems. These do not include any additional feed-forward layer (as in the *Black-box*) to keep the model simple, and to show the potential of the proposed approach.

While the *Time* architecture may be capable of learning tempo and rhythm features, the *Frequency* architecture will try to capture relevant timbral cues. Considering that these neural networks are trained using the Ballroom dataset (where the temporal cues are of relevance), we expect the former one to outperform the latter.

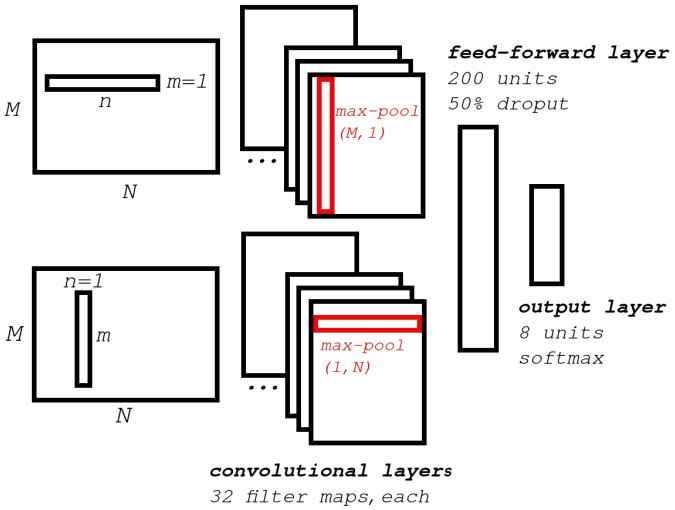


Figure 3.4: The *Time-Frequency* architecture.

Finally, we join in the same model the *Time* and *Frequency* architectures. To this end, we will select the *Time* and *Frequency* architectures that achieved better results in our preliminary experiments (see Table 3.1). Figure 3.4 depicts a schema of this combined architecture. The motivation behind this model is to join the *Time* and *Frequency* architectures, that are learning complementary aspects from the data, to create a more expressive musically motivated architecture. A feed-forward layer of 200 units is set on top of the *Time* and *Frequency* architectures to allow the model to combine time and frequency representations. We experiment with two setups, learning from: *(i)* random initialization or *(ii)* weights initialized with the previous *Time* and *Frequency* most successful models. We refere to these architectures as *(i) Time-Frequency* and *(ii) Time-Frequency (pre-initialized)*, respectively. The *Time* and *Frequency* parts in the *Time-Frequency (pre-initialized)* architecture are initialized using the best *Time* and *Frequency* models, respectively. Each initialization considers its corresponding model trained earlier in the same fold to avoid training/testing with the same data.

How to set the hyper-parameters?

Previously, we argued in favor of several design choices by rationalizing the process of deciding which CNNs might be more appropriate for the task at hand. In the following, we provide further discussion to understand the potential impact of this (intuitive) design strategy.

For example, how to set the filter length in CNNs? One of our goals is to model the relevant temporal dependencies present in the Ballroom dataset. To this end, e.g., short temporal filters may have difficulties for learning slow tempos. To study this phenomenon, we

investigate two filter lengths for the *Time* architecture:

- *60 frames (≈ 1.4 sec).* Only 1 beat can be learned by this filter for the slowest tempo in the dataset: 60 BPMs. However, the filter can learn ≈ 5 beats for the fastest tempo: 224 BPMs.

- *200 frames (≈ 4.6 sec).* Up to 4 beats can be learned by this filter at the slowest tempo in the dataset: 60 BPMs. However, the filter can fit 17 beats for the fastest tempo: 224 BPMs.

For the *Black-box* architecture we study two filter lengths: $n = 8$ and $n = 200$. The former is based on previous work where the filter shape was optimized for estimating the Ballroom classes, while the latter is motivated by the previous discussion. For experiments with $n = 8$ and $n = 60$, the input spectrogram is set to 80 frames. However, for experiments with $n = 200$, the input is enlarged to 250 frames. Note that as result of increasing the number of frames available for each spectrogram, less spectrogram segments are sampled per track.

In addition, we want to provide further evidence and discussion regarding the capacity of CNNs to learn pitch invariant representations. Intuitively, it seems beneficial to allow the filters to convolve along with the frequency axis ($m < M$). In that way, these can learn frequency features that are less pitch dependent. Consequently, the resulting filters might be capable of learning more general concepts like timbre, instead of notes. Within the *Frequency* architecture experiments, we asses several frequency filter shapes (setting m differently) — to study the performance of these filters that are capable to convolve along the frequency axis.

Table 3.1: Four architectures are studied: *Black-box*, *Time*, *Frequency* and *Time-Frequency*. Each one is presented in a block, divided by horitzontal lines. For *Black-box* and *Time*, we study different filter lengths. For *Frequency*, we study pitch invariant filters. For *Time-Frequency*, we study different initialization schemes.

Architecture	Input (M,N)	Filter shape (m,n)	# param.	Max-pool	Accuracy: mean ± std (10 folds)	Baseline
Black-box	(40,80)	(12,8)	3.275.312	(4,1)	**87.25** ± 3.39%	96%
Black-box	(40,250)	(12,200)	2.363.440	(4,1)	82.80 ± 5.12%	96%
Time	(40,80)	(1,60)	7.336	(40,1)	**81.79** ± 4.72%	82.3%
Time	(40,250)	(1,200)	19.496	(40,1)	81.52 ± 3.87%	82.3%
Frequency	(40,80)	(30,1)	3.816	(1,80)	59.45 ± 5.02%	15.9 %
Frequency	(40,80)	(32,1)	3.368	(1,80)	**59.59** ± 5.82%	15.9%
Frequency	(40,80)	(34,1)	2.920	(1,80)	58.17 ± 3.58%	15.9%
Frequency	(40,80)	(36,1)	2.472	(1,80)	57.88 ± 5.38%	15.9%
Frequency	(40,80)	(38,1)	2.024	(1,80)	57.45 ± 5.93%	15.9%
Frequency	(40,80)	(40,1)	1.576	(1,80)	52.43 ± 5.63%	15.9%
Time-Frequency	(40,80)	(1,60)-(32,1)	196.816	(40,1)-(1,80)	86.54 ± 4.29%	96%
Time-Frequency (pre-initialized)	(40,80)	(1,60)-(32,1)	196.816	(40,1)-(1,80)	**87.68** ± 4.44%	96%

Results & Discussion

In our experiments ReLUs are used as non-linearities with a final 8-way softmax, where each output unit corresponds to a Ballroom class. 50% dropout (Srivastava et al., 2014) is used to train the feed-forward layers. The output unit having the highest output activation is selected to be the model's class prediction. Each model is trained using stochastic gradient descent, with minibatches of 10 samples, minimizing the categorical cross-entropy between predictions and targets. It is trained using an initial learning rate of 0.01 and the learning rate is divided by two every time the training loss does not improve for 40 epochs. The model reporting better accuracy in the validation set is kept as the best model to be evaluated in the test set. All experiments are developed using Lasagne (a Theano-based framework that allows GPU acceleration) and are accessible online[5].

Accuracies are computed using 10-fold cross validation, with a randomly generated train-validation-test split of 80%-10%-10%. Since the input spectrograms are shorter than the total length of the song spectrogram, several estimations for each song can be done. A simple majority vote approach serves to decide the estimated class.

Results are presented in Table 3.1, where we can observe that the *Black-box* architecture reached inferior accuracy results than the state-of-the-art result by Marchand and Peeters (2016): 96%. Besides, we observe that the deep learning models can not achieve results beyond 88% accuracy. It might be that the training dataset is too small (less than 1000 tracks) for such large models (with more than 2M parameters) to learn representations that can generalize.

The *Time* architecture is capable of achieving similar results than

[5]http://github.com/jordipons/CBMI2016

Gouyon et al. (2004), who achieved an accuracy score of 82.3% with a k-nearest neighbor (with k=1) using the BPM annotations. This result provides evidence that it is valuable to first understand what's in our training datasets. By doing so, researchers can utilize such knowledge to design architectures that better fit the nature of their problem. This is especially relevant for modeling music computationally, since it has already been pointed out that machine learning algorithms are learning how to "reproduce the ground truth" rather than learning musical concepts (Sturm, 2014). Designing deep learning architectures considering musical domain knowledge may reduce that risk pointed by Sturm (2014), and will increase the capability of the systems to learn musically relevant features.

Wider filters (*n=200*) do not estimate better the Ballroom classes than *shorter* ones (*n=60*). This result is surprising because it seems a challenging task, even for a human, to discriminate tempo and rhythm with such short sounds — with less than 2 seconds. Two plausible reasons exist to explain that *shorter* filters are estimating better the Ballroom classes: *(i)* predicting the Ballroom classes does not only mean to explicitly predict tempo/rhythm, and *(ii)* less training data is available.[6] Exploring data augmentation paradigms may be interesting to improve our results with longer filters. In fact, data augmentation is a powerful tool where musically motivated choices can be done. For example, Nam et al. (2015) proposed a method called *onset-based sampling*, that samples tracks considering the onset times instead of sampling arbitrarily. However, many other musically inspired data augmentation strategies could be adopted: pitch shifting (Salamon and Bello, 2017), time stretching (Schlüter and Grill,

[6]Remember: when using longer filters we need to input longer spectrograms to the network, which decreases the number of sampled spectrograms per track.

2015) or even re-mixing (Huang et al., 2015).

Interestingly, the *Frequency* architecture is capable of learning discriminative frequency features from the data. It clearly outperforms its baseline (a naive system predicting the most likely class), denoting that the frequency features are more relevant for predicting Ballroom classes than expected. However, since this architecture was not designed to learn temporal dependencies (that are important for differentiating Ballroom classes), it does not outperform the others.

Moreover, designing the *Frequency* filters such that they can convolve in frequency ($m < M$), helps predicting the Ballroom classes. This prevents the filters to learn individual pitches centering its capacity on modeling timbre, which allows the model to be more expressive. Also, note that the performance improves until m is reduced down to 32. We speculate that smaller filters ($m < 32$) may have difficulties in learning timbral features. Finally, note that for the *Time* architecture it also exists a similar idea to the "pitch invariance" one that we discuss here. By allowing the filter to convolve along time, the network is able to fit the filter at that point in time where the beats are happening. Consequently, up to some extent, these temporal filters can be considered time-position invariant.

The last block of experiments shows that the musically motivated *Time-Frequency* architectures can achieve similar results as the *Black-box*. However, *Time-Frequency* models have two advantages: these are significantly smaller, and these are more interpretable. In essence, having a smaller model (with less learnable parameters) increases the capacity of the model to generalize with unseen data. Besides, *Time-Frequency* architectures are more interpretable since they were designed for having under control what the neural net-

work is passing through layers. Consequently, we expect these musically motivated architectures to be more treatable, as they should allow researchers to dig into what the networks have learned in a more intuitive way. Additionally, we found that the *Time-Frequency (pre-initialized)* model estimates slightly better the Ballroom classes than *Time-Frequency*, denoting that pre-initializing the model can be beneficial — it allows to start the optimization problem closer to a minimum with stronger generalization properties.

Finally, we also want to remark the importance of the training datasets for deep learning experiments. The here presented proof of concept was possible because the musical characteristics of the Ballroom dataset are well known. This allowed us to address the architectures and experiments design considering the available musical domain knowledge we had at hand. Larger datasets with musicological description are indeed necessary for the field to advance.

Interpretability

To further understand which might be the impact of using the proposed design strategy, we also visualize the learnt filters. In Figure 3.5 we depict a representative selection of those.

On the left-hand side, we observe the 12×8 filters of the *Black-box* architecture. On those, we found three main patterns: *(i)* onsets, *(ii)* harmonics, and *(iii)* others that we are not certain of what these represent. It is interesting to observe how the filters modeling onsets are clearly wasting the representational power that the vertical axis of the filter is providing, because it just repeats the onset pattern vertically. This behavior clearly denotes that the model is not employing the vertical capacity of these filters. And, on the other side, it

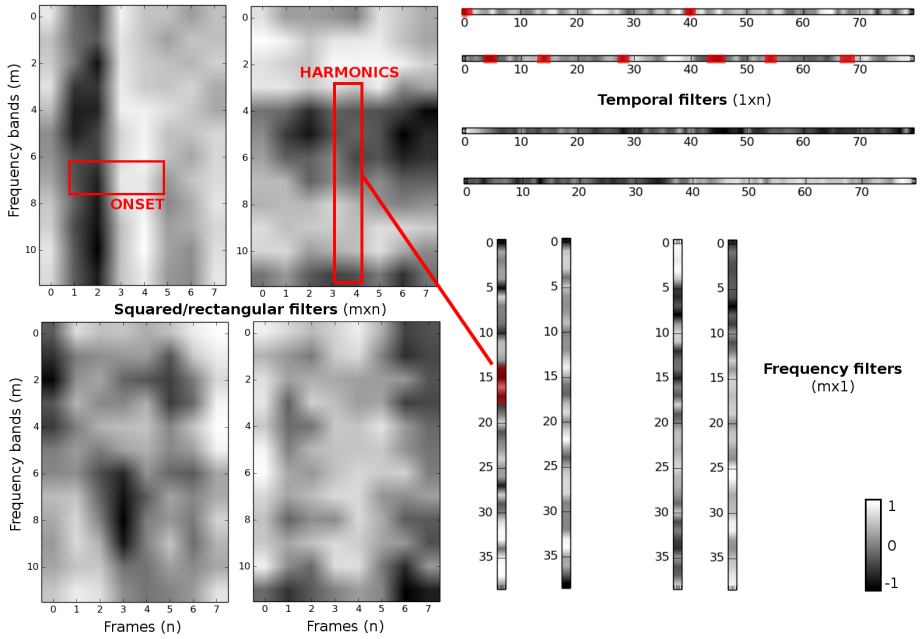


Figure 3.5: The learnt representations when using squared, vertical or horizontal CNN filters.

is also interesting to observe that the filters modeling harmonics are just representing a part of what seems important to capture timbral traces: these are modeling harmonic partials instead of capturing the whole harmonic pattern that would encode timbral information.

On the top-right side, we observe long temporal filters. We found some filters that were able to efficiently encode tempo and rhythmic patterns (first and second filters). Some others were capturing low energy components of the spectrum (third filter). And, interestingly, some others were expressing onsets (fourth filter) in a rather inefficient way — because note that with very short filters, of length between 2-10, the model should be already capable to represent onsets more efficiently (with less learnable parameters).

Finally, on the bottom-right side, we observe frequency filters. As expected, some of those are capturing timbral traces (first and second filters). For some others (like the third filter), it is not clear what these are representing. However, provided that we know that the vertical axis of the filter describes frequency content, we can interpret that this filter is emphasizing some spectral bands. Note that his kind of interpretations are harder to derive from the *Black-box* model, since its filters do not necessarily represent a musically relevant context. Finally, note that the last filter is sensible to the parts of the spectrogram having low energy in high frequencies.

Throughout this analysis, we observed that it is hard to find the appropriate filter shape so that it can efficiently represent all the musically relevant contexts present in spectrograms. Note, however, that by composing the first CNN layer with many filter shapes (instead of one single filter shape) would help to overcome this challenge. In the following, we explore this possibility as a way to capture the diverse set of musically relevant contexts that are present in music spectrograms.

Conclusions

Along this section, we studied spectrogram-based CNNs. An interesting property of these models is that the dimensions of the first-layer CNN filters are interpretable in time and frequency. We have used this observation to discuss how several filter shapes can model musically relevant contexts. By following this intuition, we proposed several musically motivated CNNs and showed that these can achieve competitive results on predicting the Ballroom dataset classes. These preliminary results show the validity of the proposed approach, which we further develop during the rest of this section.

3.3 Learning temporal representations

After showing the viability of musically motivated CNNs in Section 3.2, our goal is to further investigate the possibilities of this approach. Along this section, we focus on efficiently learning temporal representations (like tempo or rhythm) with CNNs.

While many researchers use CNNs with small rectangular filters for music-spectrograms classification (Choi et al., 2016; Han et al., 2017a), along that section *(i)* we discuss why there is no reason to use this filters' setup by default, and *(ii)* we point that more efficient architectures could be implemented if the characteristics of music are considered during the design process. Specifically, we propose a design strategy that might promote more expressive and intuitive deep learning architectures by efficiently exploiting the representational capacity of the first layer. We do so by using different filter shapes adapted to fit relevant musical contexts within the first layer.

To allow a simple comparison with our previous results, the proposed models are also trained with the Ballroom dataset.

Introduction

Due to the CNNs success in image classification, its literature significantly influenced the music informatics research community — that widely adopted standard computer vision CNNs for music classification (Choi et al., 2016; Phan et al., 2016; Han et al., 2017a). As long as these CNNs were designed for computer vision tasks, it seems reasonable that some researchers assume that audio events can be recognized by *seeing* spectrograms (since these are image-like time-frequency audio representations).

The wide use of small rectangular filters in music classification (a standard computer vision filter shape: $m \ll M$ and $n \ll N$)[7] states how straight-forward music technology researchers adopted computer vision CNNs (Choi et al., 2016; Han et al., 2017a; Schlüter and Böck, 2014). Note that image processing filter dimensions have spatial meaning while CNN-spectrogram filters dimensions correspond to time and frequency. Therefore, *wider* filters may be capable of learning longer temporal dependencies in the audio domain while *higher* filters may be capable of learning more spread timbral features. Hence, there is no strong motivation for using by default such small rectangular filters for music deep learning research since some relevant musical features (i.e.: rhythm, tempo or timbral traces) have long temporal dependencies or are spread in frequency. This observation motivates the hereby study, where we consider the characteristics of music for proposing filter shapes more suitable for music spectrograms. We hypothesize that music CNNs can benefit from a design oriented towards learning musical features rather than *seeing* spectrograms.

Moreover, we have previously pointed (in Section 3.1) that small rectangular filters can limit the representational power of the first layer since these can only represent sub-band characteristics (with a small frequency context: $m \ll M$)[7] for a short period of time (with a small time context: $n \ll N$)[7]. Hence, the network needs to combine many filters (in the same layer and/or in deeper layers) in order to model larger time/frequency contexts, what adds an extra cost to the

[7]Throughout this section we assume to use CNNs, with the input set to be music spectrograms of dimensions $M \times N$ and the CNN filter dimensions to be $m \times n$. M and m standing for the number of frequency bins and N and n for the number of time frames.

network (wider layers and/or deeper networks). Therefore, if an individual filter can model a larger time/frequency context in the first layer: *(i)* this might allow achieving a similar behavior without paying the cost of going wider and/or deeper; *(ii)* deeper layers are set free to model such context — what may allow more expressive CNNs at a similar cost since depth can then be employed for learning other features; *(iii)* filters might be more interpretable since the whole desired context would be modeled within one single filter on top of a spectrogram (with clear dimensions: time and frequency); and *(iv)* interpretable filters allows taking intuitive decisions when designing CNNs in order to make an effective use of a reduced number of learnable parameters. From previous remarks one can observe that very efficient[8] CNNs can be conceived by enabling the first layer to model larger contexts. Finally, note that given that some relevant musical features have long temporal dependencies or are spread in frequency, wide or high filters in the first layer (modeling larger time/frequency contexts) might be able to efficiently represent these features. Hence, the here proposed strategy ties very well with the previously described need of proposing musically motivated filter shapes.

Our aim is to discover novel deep learning architectures that can efficiently model music, which is a very challenging undertaking. This is why we first focus on studying how CNNs can model temporal cues, one of the most relevant music dimensions. By considering this introductory discussion, in the following, we put emphasis on the design of a single-layer CNN designed to efficiently model temporal features to later assess its accuracy in predicting the genres of the Ballroom dataset (Gouyon et al., 2004).

[8]This is the notion of efficiency we assume all through this publication.

Architectures

It is common in deep learning to model temporal dependencies in sequential data with RNNs (Böck et al., 2015; Krebs et al., 2016). Two successful methods that used RNNs for modeling temporal features from music audio are: Böck et al. (2015) for tempo estimation, and Krebs et al. (2016) for downbeat tracking. However, note that we aim to study the capacity of CNNs for modeling temporal features. Before moving forward, we want to discuss why RNNs are not considered for this study. CNNs are suitable for modeling short time-scale temporal features. By modeling short time-scale features in the first layer, deeper layers are set free for modeling other features. Hence, if a RNN layer is stacked on top of a CNN that is modeling short time-scale features (e.g., rhythm, tempo or onsets), such RNN can focus on learning *longer* time-scale temporal features. This is why we focus on the efficient modeling of short time-scale temporal features with single-layer CNNs and we leave for future work modeling *longer* time-scale temporal features with RNNs stacked on top of CNNs.

Some existing research has focused on using CNNs for modeling temporal features, proposing innovative architectures: Durand et al. (2016) used three parallel CNNs for modeling different music dimensions; Phan et al. (2016) proposed using filters representing different time-scales (setting n differently for every filter) with a max-pool layer that spans all over *time*, operation that enables time-invariance; and in Section 3.1 we proposed a light CNN for learning temporal cues with wide filters ($1 \times n$) and max-pool frequency summarization. Together with the above introduction, these works (Durand et al., 2016; Phan et al., 2016; Pons et al., 2016b) conform our conceptual basis for designing efficient CNN architectures.

Short time-scale temporal features in music audio are fundamental for describing several musically relevant concepts: *onsets* (e.g., attack-sustain-release signatures define many instruments, and these are a relevant cue for predicting genre), *rhythm* (e.g., can define a genre like waltz) or *tempo* (e.g., some genres have faster tempos than others). Note that different time-scales are required for modeling these musical concepts. For example, for modeling onsets one requires a shorter time-context than for modeling tempo or rhythm. If a long filter is used for modeling onsets, most of the weights would be set to zero: wasting part of the representational power of the filter (see Figure 3.5). Therefore, and similarly as in Phan et al. (2016), we propose setting different n's for the filters in the first layer for being able to efficiently represent several time-scale contexts.

We propose two complementary architectures meant to validate the foundations of our *design strategy*, that promotes an *efficient use of the representational capacity of the first layer by using different musically motivated filter shapes that model several (time-scale) contexts*:

(i) ***O-net*** is designed to efficiently model *onsets*, a short time-scale temporal feature. Different (short) filters of $1 \times n$ followed by a max-pool layer of $4 \times N'$ [9] might be capable of capturing in which frequency band a short time signature is occurring, with $n \in [6, 11, 16, 21, 26, 31, 36, 41]$. The *O-net* consists on 5 filters for each different filter length n. In total, there are 40 filters in the same (first) layer.

[9] *N'* and *M'* denote, in general, the dimensions of any feature map. Therefore, although the filter map dimensions will be different depending on the filter size, we will refer to their dimensions by the same name: *N'* and *M'*.

(ii) ***P-net*** is designed to efficiently model short time-scale *patterns* like rhythm or tempo. Different (longer) filters of $1 \times n$ followed by a max-pool layer of $4 \times N'$ [9] might be capable of capturing in which frequency band a time pattern is occurring, with $n \in 46 + 5 \cdot f$ where $f \in \mathbb{Z} \,|\, 0 \leq f \leq 34$ stands for the filter number. The *P-net* consists on 35 filters of different length in the first layer ranging from $46 \leq n < 216$.

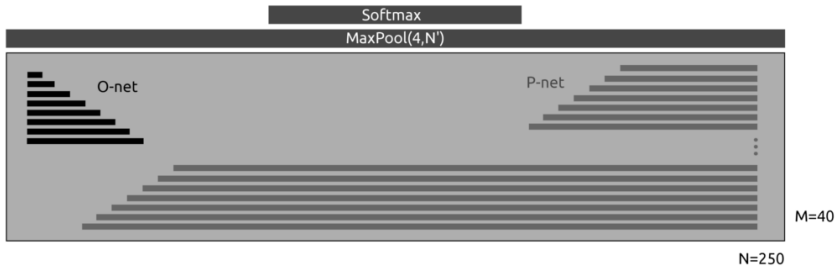


Figure 3.6: *O-net + P-net* architecture.

We propose combining these architectures in parallel as in Durand et al. (2016). As a result of that, the resulting model is shallow: a single layer with many different filters. On top of this parallel combination of CNNs (no matter which combination) we stuck a softmax layer as output. Table 1 outlines the studied models, and Figure 3.6 depicts the *O-net + P-net* architectural combination. These filters are combined by simply concatenating the resulting CNN feature maps. Consequently, one needs to carefully set the architecture so that the feature map dimensions have compatible shapes for concatenation. As a result, note that these models fulfill the specifications of our design strategy: filter shapes are intuitively designed to represent different relevant musical contexts using a reduced number of parameters in the first layer. Therefore, these models serve as test-bed to validate the proposed design strategy.

Filter and max-pool shapes discussion

We aim to investigate how CNNs can efficiently model short time-scale temporal features in music audio spectrograms. For doing so we propose using temporal filters $(1 \times n)$, that are a cheap filter expression to model temporal features where the temporal context can be easily adjusted by setting n. For example, faster patterns can be better represented by shorter filters than slower patterns, what allows minimizing the number of learnable parameters used for these filters. But also note that shorter filter lengths can facilitate modeling faster patterns, since shorter filters may better fit these patterns. Therefore, in order to efficiently model different time-scales, different filter lengths (n) are set. However, how to set the n's appropriately? We dimensioned them by defining n_O and n_P, that stand for the longer n in *O-net* and *P-net*, respectively. n_O is set to be the slowest (*longest*) onset in the dataset and n_P the slowest (*longest*) pattern. We assume 6 beats to be enough to represent a temporal pattern and therefore, the length of a *P-net* filter is determined by: $n = 1{+}5{\cdot}\Delta Fr$ where $\Delta Fr \in \mathbb{Z}$ stands for the number of frames between beats, a frame-based inter-beat interval depending on the tempo (bpm). Note that ΔFr approximates the onset length for a given tempo. Given that the slowest tempo in the dataset is of 60 bpm's (Gouyon et al., 2004) and the STFT-spectrogram is computed with a window of 2048 samples (50% overlap) at 44.1 kHz:

$$n_O \equiv \Delta Fr|_{bpm=60} = \frac{44100 \times 60_{(sec)}}{60_{(bpm)} \times 2048 \times 0.5} = 43$$

$$n_P \equiv 1 + 5 \cdot \Delta Fr|_{bpm=60} = 216$$

Note that this result corresponds with the filter lengths proposed for *O-net*: $1 \leq \Delta Fr \leq 8 \equiv 6 \leq n \leq 41 \leq n_O$ and *P-net*: $9 \leq \Delta Fr \leq 43 \equiv n_O \leq 46 \leq n \leq 216 \leq n_P$. As seen, the way we define n_O is arbitrary and depends on the dataset characteristics. Therefore, it could be that for datasets with faster tempos some patterns are learned by *O-net*. However this is not a capacity problem for the model since five filters of equal length are available in *O-net*, what enables learning onsets and patterns simultaneously (if necessary). An alternative way of seeing the design process, that would cope with the issue of *O-net* learning patterns, is to remove the distinction between onsets and patterns. However, we argue that it is interesting to define separately *O-net* and *P-net* since shorter filters are cheaper. This allows adding extra learning capacity (filters) to *O-net* at a low cost. This is why *O-net* (but not *P-net*!) includes 5 filters for each different filter length.

Additionally, note that even though $1 \times n$ filters themselves can not learn frequency features, upper layers may be capable of learning frequency cues since the frequency interpretation still holds for the resulting feature map because the convolution operation is done bin-wise (*m=1*). Actually, this observation motivates the sub-band analysis interpretation for the max-pool layer ($4 \times N'$), where the most prominent activations of the 40 bins feature map are summarized in a 10 bands feature map. Note that it is common among music technology researchers to do sub-bands analysis for modeling temporal features (Marchand and Peeters, 2014; Durand et al., 2016).

One can also note that the max-pool operation picks only the most prominent activation all over the x-axis (*N'*) of the feature map. This has two main advantages: *(i)* although the dimensionality

of the feature maps vary depending on the length of the filters, after pooling over *N'* all the feature maps have the same x-axis size – one; and *(ii)* the learnt features are time-invariant (Phan et al., 2016).

Experimental results

Experiments are realized using the Ballroom dataset that consists on 698 tracks of $\approx$ 30 sec long, divided into 8 music genres (Gouyon et al., 2004). Two main shortcomings are regularly issued against this dataset: *(i)* its small size and *(ii)* the fact that its classes are highly correlated with tempo – although being proposed for evaluating rhythmic descriptors. And precisely, the previously described shortcomings motivate our study. Deep learning approaches rely on the assumption that *large* amounts of training data are available to train the *large* number of parameters of a network, and the data assumption does not hold for most music problems. We want to study if a CNN architecture designed to efficiently represent musical concepts can achieve competitive results in a context where a *small* amount of parameters is trained from a *small* dataset. The Ballroom dataset provides an excellent opportunity for studying so, due to its reduced size and because its classes are highly correlated with short time-scale temporal features (tempo and rhythm). We exploit this *prior* knowledge to propose and assess some *small* efficient musically motivated architectures that might be capable of learning these temporal features.

The audio is fed to the network through fixed-length mel spectrograms, $N = 250$ frames wide. It is set to 250 in order to fit the longest filter in *P-net*: $n = 216$. Throughout this work we use 40 bands mel-spectrograms derived from a STFT-spectrogram com-

puted with a Blackman Harris window of 2048 samples (50% overlap) at 44.1 kHz. Phases are discarded. A dynamic range compression is applied to the input spectrograms element-wise: $log(1 + C \cdot x)$ where $C = 10.000$ (Dieleman and Schrauwen, 2014). The resulting spectrograms are normalized so that the whole dataset spectrograms (together) have zero mean and variance one. The activation functions are linear rectifiers (ReLUs) with a final 8-way softmax, where each output unit corresponds to a Ballroom class. 50% dropout is applied to the output layer. The output unit having the highest output activation is selected to be the model's class prediction. Each network is trained using gradient descent with a minibatch size of 50, minimizing the categorical cross-entropy. Networks are trained from random initialization (with the same random seed) using an initial learning rate of 0'01. A learning schedule is programmed: the learning rate is divided by ten every time the training loss gets stacked until there is no more improvement. The best model in the validation set is kept for testing. Mean accuracies are computed using 10-fold cross validation with the same randomly generated train-validation-test split of 80%-10%-10%. Since the input spectrograms are shorter than the total length of the song spectrogram, several estimations per each song can be done. A simple majority vote approach serves to decide the estimated class for each song.

We cut the input spectrograms with overlapping, and the hop-size is set differently depending on the experiment. Note that employing overlapping input spectrograms can be seen as a data augmentation technique. But also note that the smaller the hop size, the more estimations per song are done at test time – what can be useful for the majority vote stage.

Results are compared with the state-of-the-art of the Ballroom dataset (Marchand and Peeters, 2016), and against our best deep learning-based results from Section 3.1: *time*, *time-freq* and *black-box*. Marchand and Peeters (2016) is based on a scale and shift invariant time/frequency representation that uses auditory statistics, not deep learning. The *time* architecture has a single CNN layer with 1×60 filters (one filter shape in a single-layer CNN). *Time-freq* and *black-box* have two layers: CNN + feed-forward and they differ in the filter shape setup of the CNN layer. *Time-freq* uses 1×60 and 32×1 filters (two different filter shapes in the CNN layer) and *black-box* uses small rectangular filters (one filter shape in the CNN layer). For fair comparison against these models, two hop sizes are used: $hop = 250, 80$. When setting $hop = N = 250$, no input spectrograms overlap is used — equally as in Section 3.1. However, the input spectrogram is set smaller ($N = 80$) for the three previous deep learning approaches and therefore, more training examples are available. In order to have as many training examples as in Section 3.1, we also compare our results when $hop = 80$ (although overlapping data is used). These two setups ($hop = 250, 80$) provide a fair test-bed to compare our results with the state-of-the-art.

Results are presented in Table 1. First, observe that for $hop = 80$ most of the here presented models (very small and shallow) can achieve better performance than *time*, *time-freq* and *black-box*. Note that the proposed models have a single CNN layer, and a large variety of filter shapes in the first layer. But also observe that *O-net* + *P-net* architectures, having the most diverse combination of filter shapes, are the best among the presented models. Therefore, these results validate the here proposed design strategy of adding musically

Table 3.2: Mean accuracy results comparing different approaches predicting the Ballroom dataset classes. *# params* stands for the number of learnable parameters of the model and *hop* for the hop-size when cutting the input spectrograms with overlapping.

Model:	**hop**	**#params**	**accuracy**
O-net	250/80	4,188	76.66/85.24%
P-net	250/80	7,428	83.95/89.26%
2x O-net	250/80	8,368	81.53/86.54%
2x P-net	250/80	14,848	85.67/89.11%
O-net + P-net	250/80	11,608	87.25/89.68%
2x O-net + 2x P-net	250/80	23,208	87.25/91.27%
4x O-net + 4x P-net	250/80	46,408	88.82/91.55%
8x O-net + 8x P-net	250/80	92,808	88.68/**92.27**%
Marchand and Peeters (2016)	-	-	**96**%
Time	80	7,336	81.79%
Time-freq	80	196,816	87.68%
Black-box	80	3,275,312	87.25%

motivated filters with different shapes (instead of having the same filter repeated many times) in the first layer. Second, observe that this design strategy is especially useful in circumstances when not many training examples are available. Note that for $hop = 250$ bigger accuracy gains are achieved when more different filter shapes are available (for example, compare *O-net + P-net* and *2x P-net*). Since adding different filter shapes is cheaper than doubling the capacity of the network, the here proposed design strategy allows increasing the representational power of the first layer at a very low cost. Efficiently using a reduced number of parameters ($92{,}808 \ll 196{,}816 \ll 3{,}275{,}312$) for modeling the main dimensions of a problem is a straight-forward way of fighting overfitting in scenarios where *small*

datasets and little computational resources are available. Third, note that models containing *P-net* are the most successful ones. We speculate that this is because the most relevant features in this dataset (rhythm and tempo) can be better encoded in *P-net* than in *O-net*, as we hypothesized during the design process. And fourth, none of the proposed models overcome the result by Marchand and Peeters (2014). However, note the limitations of the here proposed models (basically designed to model short time-scale temporal features and to validate the proposed design strategy): only a limited amount of $1 \times n$ filters are used, within a single layer and with a limited amount of data. In future work, we plan to increase the representational capacity of the first layer by also using filter shapes designed to capture timbral traces, we want to stack more layers on top this efficient CNN, and we want to use more data to train our models.

Conclusions

We have presented a CNNs design strategy that consists on modeling different (time-scale) contexts within the first layer with different (musically motivated) filter shapes that are intuitively designed to represent (musical) concepts. Our results show that this design strategy is useful for fully exploiting the representational power of the first CNN layer for learning temporal representations. These results provide an advance in: gaining intuition towards what CNNs learn, efficiently adapting deep learning for music, and designing networks at a lower cost.

3.4 Learning timbral representations

After studying how to learn temporal representations with CNNs, in this section we study how to tailor CNNs towards learning timbral representations from log-mel magnitude spectrograms.

To this end, we first identify which are the main trends when designing CNNs. Along with this literature review we discuss what might be crucial for efficiently learning timbre representations when designing CNNs. From this discussion, we propose a design strategy meant to capture the relevant time-frequency contexts for learning timbre. Importantly, one of our goals is to design CNN architectures that can *efficiently* capture timbral traces. By doing so, we aim to reduce the number of learnable parameters of the proposed CNNs, so that the over-fitting risk of these neural networks can be reduced. Several architectures based on the design principles we propose are successfully assessed for different research tasks related to timbre: singing voice phoneme classification, musical instrument recognition and music auto-tagging. The proposed architectures can achieve equivalent results (if not better) to the state-of-the-art while significantly reducing the number of learnable parameters of the model.

Introduction

Our goal is to discover deep learning architectures that can efficiently model music signals. After showing in Section 3.3 that it is possible to design efficient CNNs for modeling temporal music features (like tempo and rhythm), we now focus on studying how to efficiently learn timbral[10] representations, one of the most salient musical features.

[10]In the following lines, we will provide a formal definition of timbre.

Historically, music audio processing for timbre description has been divided into two groups: *(i) bag-of-frames* methods and *(ii)* methods based on *temporal modeling*. On the one hand, *bag-of-frames* methods have been shown to be limited as they just model the statistical distribution of frequency content on a frame basis (Porter et al., 2015). On the other hand, methods based on *temporal modeling* consider the temporal evolution of frame-based descriptors (Rabiner, 1989; Roebel et al., 2015). Some of these methods are capable of representing spectro-temporal patterns that can model the temporal evolution of timbre (Roebel et al., 2015) and then, for example, attack-sustain-release patterns can be jointly represented.

Most previous methodologies —either based on *(i)* or *(ii)*— require a dual pipeline: first, descriptors need to be extracted using a pre-defined algorithm and parameters; and second, (temporal) models require an additional framework tied on top of the proposed descriptors. Therefore, descriptors and (temporal) models are typically not jointly designed. Throughout this chapter, we explore modeling timbre with a deep learning architecture having its input set to be magnitude spectrograms. This *quasi* end-to-end learning approach allows minimizing the effect of the fixed pre-processing steps described above. Note that no strong assumptions over the descriptors are required since a generic perceptually-based pre-processing is used: log-mel magnitude spectrograms. Besides, deep learning can be interpreted as a temporal model (if more than one frame is input to the model). In that way, the model might be capable to learn spectro-temporal descriptors from spectrograms. In this case, the (learned) descriptors and the temporal model are jointly optimized, which might imply an advantage when compared to previous methods.

From the different deep learning architectures available, we focus on CNNs due to several reasons: *(i)* by taking spectrograms as input, one can interpret the first-layer filter dimensions in the time-frequency domain; and *(ii)* CNNs can efficiently exploit musically relevant invariances, such as the time and frequency invariances present in spectrograms, by sharing a reduced amount of parameters.

Two main trends exist in the literature for modeling timbre using spectrogram-based CNNs: using *small-rectangular filters* ($m \ll M$ and $n \ll N$)[11], or using *high filters* ($m \leq M$ and $n \ll N$)[11].

- *Small-rectangular filters* (Choi et al., 2016; Han et al., 2017a) inquire the risk of limiting the representational power of the first layer since these filters are typically *too small* for modeling spread spectro-temporal patterns (see Section 3.1). Since these filters can only represent sub-band characteristics (with a small frequency context: $m \ll M$) for a short period of time (with a small time context: $n \ll N$), these can only learn, for example: onsets or bass notes (Pons et al., 2016b; Choi et al., 2015). But these filters cannot learn the cymbals' or snare-drums' time-frequency patterns in the *first layer* — since such a spread context can not fit inside a small-rectangular filter. To capture this larger context one needs to pay the computational cost of going deeper.

- Although *high filters* (Dieleman and Schrauwen, 2014; Lee et al., 2009) can fit most spectral envelopes, these require many parameters to be learnt from (typically small) data — risking to *over-fit* and/or to *fit noise*. See Figure 3.7 (right) for two examples of filters fitting noise as a result of having more context available than the required.

[11] CNNs input is set to be log-mel spectrograms of dimensions $M \times N$ and the CNN filter dimensions to be $m \times n$. M and m standing for the number of frequency bins and N and n for the number of time frames.

For example: *filter 1* is fitting noise along the frequency axis because the onset pattern is repeated all over the vertical axis; or *filter 2* is fitting noise along the time axis because the harmonic pattern is repeated all over the horizontal axis.

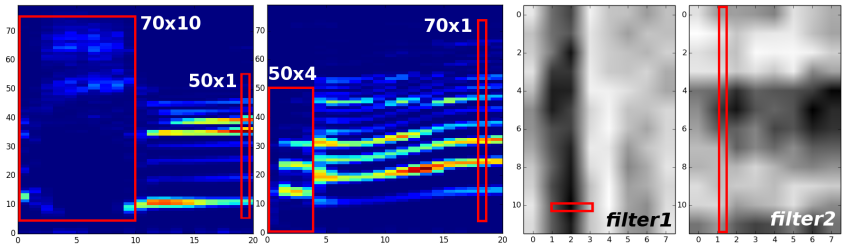


Figure 3.7: Left: two spectrograms of different sounds used for the singing voice phoneme classification experiment. **Right**: two trained small-rectangular filters of size 12×8 fitting noise to model onsets and harmonic partials, respectively. Relevant time-frequency contexts are highlighted in red.

Moreover, note that most CNN architectures use unique filter shapes in every layer (Choi et al., 2016; Dieleman and Schrauwen, 2014; Han et al., 2017a). However, recent works point out that using different filter shapes in each layer is an efficient way to exploit the CNN's capacity (Pons and Serra, 2017; Phan et al., 2016). Note that in Section 3.3 we proposed to use different musically motivated filter shapes in the first layer to efficiently model several musically relevant time-scales for learning temporal features. In the following, we propose a novel approach to this design strategy which facilitates learning musically relevant time-frequency contexts while minimizing the risk of noise-fitting and over-fitting for timbre analysis. Out of this design strategy, several CNN models are proposed and assessed for three research tasks related to timbre: singing voice phoneme classification, musical instrument recognition and music auto-tagging.

CNNs design strategy for timbre analysis

Timbre is considered as the *"color"* or the *"quality"* of a sound (Wessel, 1979). It has been found to be related to the spectral envelope shape and to the time variation of spectral content (Peeters et al., 2011). Therefore, it is reasonable to assume timbre to be a time-frequency expression and then, magnitude spectrograms are an adequate input for CNNs aiming to capture timbral traces. Although phases could be used, these are commonly discarded (Choi et al., 2016; Dieleman and Schrauwen, 2014; Han et al., 2017a). Moreover, timbre is often defined by what it is not: *"a set of auditory attributes of sound events in addition to pitch, loudness, duration, and spatial position"* (McAdams, 2013). In concordance with that definition, we propose ways to design CNN filters invariant to these attributes:

- ***Pitch invariance***. By enabling filters to convolve along with the frequency domain of a mel spectrogram (f_0 shifting), the resulting filter and feature map can represent timbre and pitch information separately. However, if filters do not capture the whole spectral envelope encoding timbre (because these model a small frequency context), previous discussion does not necessarily hold. Additionally, depending on the used spectrogram representation (e.g., STFT, or mel) CNN filters might learn more robust pitch invariant features. Note that STFT timbre patterns are f_0 dependent. However, mel timbre patterns are more pitch invariant than STFT ones because these are based in a different (perceptual) frequency scale. Besides, one can achieve further pitch invariance if a max-pool layer spanning all over the vertical axis (M')[12] is applied to the resulting CNN feature map.

[12] N' and M' denote, in general, the horizontal and vertical dimensions of any feature map. Therefore, although the filter map dimensions will be different

- ***Loudness invariance*** for CNN filters can be approached by using weight decay — L2-norm regularization of filter weights. By doing so, filters are normalized to have low energy and energy is then expressed into feature maps. Loudness is a perceptual term that we assume to be correlated with energy.

- ***Duration invariance***. Firstly, $m \times 1$ filters are time-invariant by definition since these do not capture duration. Temporal evolution is then represented in the feature maps. Secondly, sounds with determined length and temporal structure (like kick drums or cymbals) can be well captured with $m \times n$ filters. These are also duration invariant because such sounds last a fixed amount of time. Note the resemblance between the first layer $m \times 1$ filters with the frame-based descriptors; and between the first layer $m \times n$ filters with the spectro-temporal descriptors.

- ***Spatial position invariance*** is achieved by down-mixing (for example by averaging all channels) whenever the audio is not mono.

From previous discussion, one can easily notice the importance the CNN filter shapes of the first layer. They play a crucial role for defining pitch invariant and duration invariant CNNs. For that reason, we propose to use our domain expertise for deciding the filter shapes. For example, by visually inspecting Figure 3.7 (left) one can easily detect which are the relevant time-frequency contexts in these spectrograms: frequency $\in [50, 70]$ and time $\in [1, 10]$. These measurements provide intuitive guidance when designing the filter shapes for the first CNN layer — in this case, for the task of singing voice phoneme classification.

depending on the filter size, we refer to these dimensions by the same name: *N'* and *M'*, respectively.

Finally, we discuss how to efficiently learn timbre features with CNNs. Timbre is typically expressed at different scales in spectrograms. For example, cymbals are more spread in frequency than bass notes, or vowels typically last longer than consonants in singing voice. If a unique filter shape is used within a layer, one can inquire the risk of: *(a)* fitting noise because too much context is modeled and/or *(b)* not modeling enough context.

- *Risk (a)*. Figure 3.7 (right) depicts two filters that have fit noise. Observe that *filter1* is repeating a noisy copy of an onset throughout the frequency axis, and *filter2* is repeating a noisy copy of three harmonic partials throughout the temporal axis. Note that much more efficient representations of these musical concepts can be achieved by using different filter shapes: 1×3 and 12×1, respectively (in red). Using the adequate filter shape allows minimizing the risk to fit noise and also minimizes the risk to over-fit the training set (because the CNN model size is also reduced).

- *Risk (b)*. The frequency context of *filter2* is too small to model the whole harmonic spectral envelope, and it can only learn three harmonic partials — what is limiting the representational power of this (first) layer. A straightforward solution for this problem is to increase the frequency context of the filter. However, note that if we increase it too much, such filter is more prone to fit noise.

Using different filter shapes allows reaching a compromise between risks *(a)* and *(b)*. Consequently, using different filter shapes within the first layer seems crucial to efficiently model different musically relevant time-frequency contexts with spectrogram-based CNNs. Moreover, this design strategy ties very well with the idea of using the available domain knowledge for designing filter shapes. Domain knowl-

edge can intuitively guide the different filter shapes design so that spectro-temporal envelopes can be efficiently represented within a single layer. Interestingly, though, another possible solution exists to capture these musically relevant contexts (or receptive fields) in deep learning: to combine several small filters (either in the same layer or by going deep) until the desired context is represented. However, several reasons exist for supporting the here proposed approach: *(i)* the principle by Hebb (1949) from neuroscience (cells that fire together, wire together), and *(ii)* learning complete spectro-temporal patterns within a single filter allows to inspect and interpret the learnt filters in a compact way.

The above discussion introduces the fundamentals of the proposed design strategy for timbre analysis with CNNs. In the following, we run a set of experiments to assess the capacities of the proposed design strategy. In particular, we investigate how these perform for three research tasks related to timbre: singing voice phoneme classification, musical instrument recognition and music auto-tagging.

Experiments

The audio is fed to the neural network using fixed-length log-mel spectrograms. Phases are discarded. Spectrograms are normalized: zero mean and variance one. Activation functions are ELUs (Clevert et al., 2016). Architectures are designed according to the proposed strategy we introduced in previous discussion — by employing: weight decay regularization, monaural signals, and different filter shapes in the first layer. Each network is trained to optimize the cross-entropy loss with SGD from random initialization (He et al., 2015). The best model in the validation set is kept for testing.

Singing voice phoneme classification

The jingu[13] a cappella singing audio dataset by Black et al. (2014) is annotated with 32 phoneme classes[14] and consists of two different role-types of singing: *dan* (young woman) and *laosheng* (old man). The *dan* part has 42 recordings (89 minutes) and comes from 7 singers; the *laosheng* part has 23 arias (39 minutes) and comes from other 7 *laosheng* singers. Since the timbral characteristics of *dan* and *laosheng* are very different, the dataset is divided into two. Each part is then randomly split —train (60%), validation (20%) and test (20%)— for assessing the presented models for the phoneme classification task. Audio was sampled at 44.1 kHz. STFT was performed using a 25ms window (2048 samples with zero-padding) with a hop size of 10ms. This experiment assesses the feasibility of taking architectural decisions based on domain knowledge for an efficient use of the network's capacity in small data scenarios. The goal is to do efficient deep learning by taking advantage of the design strategy we propose. This experiment is specially relevant because, in general, no large annotated music datasets are available — this dataset is an example of this fact. The proposed architecture has a single wide convolutional layer with filters of various sizes. The input is of size 80×21. The CNN makes a decision for a frame given its context: ±10ms, 21 frames in total. Considering the above discussion, we use 128 filters of sizes 50×1 and 70×1, 64 filters of sizes 50×5 and 70×5, and 32 filters of sizes 50×10 and 70×10. A max-pool layer of $2\times N'$ follows before the 32-way softmax output layer with 30% dropout.

[13] *"Jingu"* is also known as *"Beijing opera"* or *"Peking opera"*.

[14] Annotation and more details can be found in:
https://github.com/MTG/jingjuPhonemeAnnotation

MP(2,N') was chosen to achieve time-invariant representations while keeping the frequency resolution.

We use overall classification accuracy as evaluation metric and results are presented in Table 3.3. As a baseline, we also train a 40-component Gaussian Mixture Models (GMMs), a fully-connected MLP with 2 hidden layers (MLP) and Choi et al. (2016) architecture, that is a 5-layer CNN with small-rectangular filters of size 3×3 (Small-rectangular). All the architectures are adapted to have a similar amount of parameters (so that results are comparable). GMMs features are: 13 coefficients MFCCs, their deltas and delta-deltas. 80×21 log-mel spectrograms are used as input for the other competing models. Implementations are available online[15].

Table 3.3: The models' performance for *dan* & *laosheng* datasets.

	dan / #param	*laosheng* / #param
Proposed	**0.484** / 222k	**0.432** / 222k
Small-rectangular	0.374 / 222k	0.359 / 222k
GMMs	0.290 / -	0.322 / -
MLP	0.284 / 481k	0.282 / 430k

The proposed architecture outperforms the other models by a significant margin (although being a single-layered model), what denotes the potential of the proposed design strategy. Deep models based on small-rectangular filters —which are state-of-the-art in other music-classification tasks (Choi et al., 2016; Han et al., 2017a)— do not perform as well as the proposed model for these small datasets. As future work, we plan to investigate deeper models that can take advantage of the richer set of features learnt by the proposed model.

[15] https://github.com/ronggong/EUSIPCO2017

Musical instrument recognition

The IRMAS dataset (Bosch et al., 2012) training split contains 6705 audio excerpts of 3 seconds length labeled with a single predominant instrument. Testing split contains 2874 audio excerpts of length 5∼20 seconds labeled with more than one predominant instrument. 11 pitched class instruments are annotated. Audios are sampled at 44.1kHz. The state-of-the-art for this dataset corresponds to a deep CNN based on small-rectangular filters (of size 3×3) by Han et al. (2017a). Moreover, another baseline is provided based on a standard bag-of-frames approach + SVM classifier proposed by Bosch et al. (2012). We experiment with two architectures based on the proposed design strategy:

- *Single-layer* has a single but wide CNN layer with filters of various sizes. The input is set to be of size 96×128. We use 128 filters of sizes 5×1 and 80×1, 64 filters of sizes 5×3 and 80×3, and 32 filters of sizes 5×5 and 80×5. We also max-pool the *M'* dimension to learn pitch invariant representations: MP(*M',16*). 50% dropout is applied to the 11-way softmax output layer.

- *Multi-layer* architecture's first layer has the same settings as *single-layer* but it is deepened by two convolutional layers of 128 filters of size 3×3, one fully-connected layer of size 256 and a 11-way softmax output layer. 50% dropout is applied to all the dense layers and 25% for convolutional layers. Each convolutional layer is followed by max-pooling, where the first wide layer is of *MP(12,16)* and deeper layers are of *MP(2,2)*.

Implementations are available online[16]. The STFT is computed using

[16] https://github.com/Veleslavia/EUSIPCO2017

512 points FFT with a hop size of 256. Audios were down-sampled to 12kHz. Each convolutional layer is followed by batch normalization (Ioffe and Szegedy, 2015). All convolutions use *same* padding. Therefore, the dimensions of the feature maps out of the first convolutional layer are still equivalent to the input (time and frequency). Then, the resulting feature map of the *MP(12,16)* layer can be interpreted as an eight-bands summary (96/12=8). This max-pool layer was designed considering: *(i)* is relevant to know in which band a given filter shape is mostly activated — as a proxy for knowing in which pitch range timbre is occurring; and *(ii)* is not so relevant to know when it is mostly activated. To obtain instrument predictions from the softmax layer we use the same strategy as Han et al. (2017a): estimations for the same song are averaged and then a threshold of 0.2 is applied. In Table 3.4, we report the standard metrics for this dataset: micro- and macro- precision, recall and f1 score. The micro-metrics are calculated globally for all testing examples while the macro-metrics are calculated label-wise and the unweighted average is reported.

Table 3.4: Recognition performance for the IRMAS dataset.

	Micro			Macro		
Model / #param	P	R	F1	P	R	F1
Bosch *et al.*	0.504	0.501	0.503	0.41	0.455	0.432
Han *et al.* / 1446k	**0.655**	**0.557**	**0.602**	0.541	0.508	0.503
Single-layer / 62k	0.611	0.516	0.559	0.523	0.480	0.484
Multi-layer / 743k	0.650	0.538	0.589	**0.550**	**0.525**	**0.516**

Multi-layer achieved similar results as the state-of-the-art with twice fewer *#param*. This result denotes how efficient are the proposed architectures. Moreover, note that small filters are also used within the proposed architecture. We found these filters to be im-

portant for achieving state-of-the-art performance – although no instruments with such *small* time-frequency signature (such as kick drum sounds or bass notes) are present in the dataset. However, if such small filters are substituted with long-vertical filters, the performance does not drop dramatically. Finally note that *single-layer* still achieves remarkable results: it outperforms the standard bag-of-frames + SVM approach.

Music auto-tagging

Automatic tagging is a multi-label classification task. We approach this problem with the MagnaTagATune dataset (Law et al., 2009), with 25.856 clips of $\approx$ 30 seconds sampled at 16kHz. Predicting the top-50 tags of this dataset (instruments, genres and others) has been a popular benchmark for comparing deep learning architectures. Architectures from Choi et al. (2016) and Dieleman and Schrauwen (2014) are set as baselines — that are state-of-the-art examples of architectures based on small-rectangular filters and high filters, respectively. Therefore, this dataset provides a nice opportunity to explore the tradeoff between *(i)* leaning little context in the first layer with small-rectangular filters, and *(ii)* risking to fit noise with high filters. Choi et al. (2016) architecture consists of a CNN of five layers where filters are of size 3×3 with an input of size 96×187. After every CNN layer, batch normalization and max-pool is applied. Dieleman and Schrauwen (2014) architecture has two CNN layers with filters of M×8 and M'×8 size, respectively. The input is of size 128×187. After every CNN layer a max-pool layer of 1×4 is applied. Later, the penultimate layer is a fully connected layer of 100 units. An additional baseline is provided: *Small-rectangular*, which is an adaption

of Choi et al. (2016) architecture to have the same input and number of parameters as Dieleman and Schrauwen (2014). All models use a 50-way sigmoidal output layer and the STFT was performed using 512 points FFT with a hop size of 256.

Our experiments reproduce the same conditions as in Dieleman and Schrauwen (2014), since the proposed model adapts their architecture to the proposed design strategy. We uniquely modify the first layer to have many musically motivated filter shapes. Other layers are kept intact. This allows isolating our experiments from confounding factors, so that we uniquely measure the impact of increasing the representational capacity of the first layer. Inputs are set to be of size 128×187. Since input spectrograms (≈3 seconds) are shorter than the total length of the song, estimations for the same song are averaged. We consider the following frequency contexts as relevant for this dataset: *m=100* and *m=75* to capture different wide spectral shapes (like genres timbre, guitar or cello), and *m=25* to capture shallow spectral shapes (like drums). For consistency with Dieleman and Schrauwen (2014), we consider the following temporal context: *n=[1,3,5,7]*. We use several filters per shape in the first layer:

- *m=100*: 10x 100×1, 6x 100×3, 3x 100×5 and 3x 100×7.
- *m=75*: 15x 75×1, 10x 75×3, 5x 75×5 and 5x 75×7.
- *m=25*: 15x 25×1, 10x 25×3, 5x 25×5 and 5x 25×7.

For merging the resulting feature maps, these need to be of the same dimension. We zero-pad the temporal dimension before the first layer convolutions and we use max-pool layers of size *MP(M',4)*. Note that all the resulting feature maps have the same dimension: $1 \times N'$, and are pitch invariant. 50% dropout is applied to all dense layers. We also evaluate variants of the proposed model where the

number of filters per shape in the first layer are increased according to a factor. The rest of the layers are kept intact. Implementations are available online[17].

Table 3.5: The models' performance for the MTT dataset.

Model	AUC/#param	Model	AUC/#param
Small-rectangular	86.52 / 75k	Choi *et al.*	**89.40** / 22M[18]
Dieleman *et al.*	88.15 / 75k	Proposed x2	**89.33** / 191k
Proposed	88.95 / 75k	Proposed x4	88.71 / 565k

We use area under the ROC curve (AUC) as a metric for our experiments. Table 3.5 (left column) shows the results of three different architectures with the same number of parameters. The proposed model outperforms the others, denoting that architectures based on the design strategy we propose can better use the capacity of the CNN. Moreover, Table 3.5 (right column) shows that it is beneficial to increase the representational capacity of the first layer — up to the point where we achieve equivalent results to the state-of-the-art while significantly reducing the number of learnable parameters (*#param*) of the model.

Conclusions

Inspired by the fact that it is hard to identify the adequate combination of parameters for a deep learning model —what leads to architectures being difficult to interpret—, we decided to incorporate domain knowledge during the architectural design process. This lead us to discuss some common practices when designing CNNs for music

[17] https://github.com/jordipons/EUSIPCO2017

[18] Although equivalent results can be achieved with 750k parameters.

classification, with a specific focus on how to efficiently learn timbre representations. This discussion motivated the design strategy we present for modeling timbre using spectroram-based CNNs. Several ideas were proposed to achieve pitch, loudness, duration and spatial position invariance with CNNs. Moreover, we proposed actions to increase the efficiency of these models. The idea is to use different filter shapes in the first layer that are motivated by audio domain knowledge. Namely, we propose to use different musically motivated filter shapes in the first layer. Besides providing theoretical discussion and background for the proposed design strategy, we also validated it empirically. Several experiments in three datasets for different tasks related to timbre (singing voice phoneme classification, musical instrument recognition and music auto-tagging) provide empirical evidence that this approach is powerful and promising. In these experiments, we evaluate several architectures based on the presented design strategy to show that the proposed design strategy is effective in all cases. These results support the idea that increasing the representational capacity of the layers can be achieved by using different filter shapes. Specifically, the proposed architectures used several filter shapes having the capacity of capturing timbre with *high enough* filters. For example, we found very remarkable the results of the single-layered models we proposed. Since single-layer architectures use a reduced amount of parameters, these might be very useful in scenarios where small data and a few hardware resources are available. Furthermore, when deepening the network we were able to achieve equivalent results to the state-of-the-art — if not better.

3.5 Summary and conclusions

Our aim is to efficiently model music audio with deep neural networks, which is a very challenging undertaking. Some of these challenges are that music exhibits long-term dependencies (that define the tempo or the rhythm of a song), or that it is difficult to define what's timbre in music. Can we design deep learning architectures to efficiently capture these musically relevant characteristics?

As seen in this chapter, it exists no consensus on which are the best deep learning architectures to fit music audio — because it is hard to discover the adequate combination of hyper-parameters for a particular task, which can lead to architectures being difficult to interpret. As as en example of this, note that many computer vision architectures are used for processing spectrograms like images (Choi et al., 2016). However, images have a spatial meaning — while the spectrograms' axis stand for time and frequency. With this in mind, our goal is to rationalize this design process by exploring deep learning architectures specifically thought to fit music audio — so that these can be more successful and understandable.

To this end, we have presented a novel CNNs design strategy that consists of modeling different (time-frequency) contexts within the first CNN layer with different (musically motivated) filter shapes that are intuitively designed to represent musical characteristics. Throughout that chapter have shown that by following this design strategy, it is possible to efficiently learn temporal and timbral representations. For example, we employ different "long" filter shapes in the first CNN layer to efficiently capture the tempo and rhythm features present in music spectrograms. In addition, we have also studied

how to capture timbral traces with spectroram-based CNNs. Several ideas were proposed to achieve pitch, loudness, duration and spatial position invariance with CNNs. Our experimental results show that by following this design strategy, we can design very compact CNNs that are interpretable and perform competently. We show that musically motivated CNNs can achieve results that are comparable (if not better) to the state-of-the-art for several tasks.

To conclude, we want to remark that the here proposed design strategy heavily relies on domain knowledge. It is important to note that this strategy might be constraining the solution space of the proposed deep learning models. From one side, this can be advantageous. By restricting the solution space (with architectures specifically tailored towards modeling musical features), we expect these deep learning models to have more generalization capabilities and to be more interpretable. On the other side, one inquires the risk of not allowing the model to be expressive enough[19] — what might be of utility in scenarios where not much training data are available, but it can be limiting when enough data and compute power are accessible.

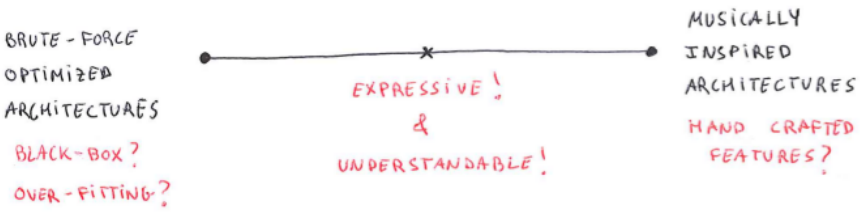


Figure 3.8: Graphic representation of the trade-off between pure data-driven approaches and pure knowledge-based approaches.

[19] In other words, that the model could be over-regularized, or over-constrained.

In the following lines, we discuss a trade-off that motivated some of our work when investigating which deep learning architectures are most suitable for music audio signals. This discussion departs from the idea that it is unreasonable to assume that sizable amounts of training data are generally accessible to train deep neural networks. Hence, regularizing the solution space of these models can be an interesting direction to peruse when not much training data is available. From this perspective, the trade-off we discuss can be seen as a way to utilize domain knowledge for regularizing the solution space.

Above, we identified a trade-off between pure data-driven approaches and pure knowledge-based approaches. On the left-hand side of Figure 3.8, we represent the non-constrained models that explore the solution space by any means: brute-force optimization of hyper-parameters and architectures, a pure data-driven approach that is learning from data. Part of the deep learning game is to allow the architecture to freely discover features, which leads to very successful models. However, a common criticism to deep learning relates with the difficulty in understanding the underlying relationships that the neural networks are learning, thus behaving like a black-box. Having more interpretable models can be valuable since it has already been pointed that machine learning algorithms are learning how to "reproduce the ground truth" rather than learning musically relevant characteristics (Sturm, 2014). If these models explore the solution space without any constraint, it exists the risk of delivering a solution difficult to interpret that, besides, can be prone to over-fitting — given that a small sample of data is generally available for training.

On the right-hand side of Figure 3.8, handcrafted features and knowledge-based approaches are represented. This family of mod-

els rely on domain knowledge to restrict the solution space so that these have more chances to generalize although few training data are available. One can think of that as a domain-knowledge informed inductive bias. However, restricting too much the solution space of deep neural networks can be problematic because the learning algorithm won't be able to sufficiently explore the solution landscape. Hence, if deep neural networks are not expressive enough (e.g., due to the architecture, initialization schema, or input representation), such models are likely to perform poorly. In other words, by restricting the solution space we are inquiring the risk of not fully exploiting the power of deep learning.

Therefore, provided that training data are generally scarce, it might be interesting to explore a compromise between *a)* expressiveness, and *b)* domain-knowledge informed constraints in deep neural networks. In this chapter, our goal was to constrain the solution space in a way that allows model interpretability while still guaranteeing the expressiveness of deep neural networks. If the solution space is severely limited by the intervention of the designer (as it can be currently happening with hand-crafted features and knowledge-based classifiers), we might be missing the opportunity that deep learning is offering to train highly expressive models. That said, the deep learning framework allows a compromise between the two previously described model paradigms. Deep learning can incorporate domain knowledge in several parts of its pipeline: in its input, architecture, cost, or weights initialization — what can tailor the model towards learning solutions closer to what is expected. For example, by designing musically inspired deep learning architectures, one aims to place the model close to a plausible solution having the generaliza-

tion power of the known priors that are encoded in the architecture.

Throughout this chapter, we have investigated to constrain the solution space by employing musically motivated deep learning architectures together with some minimal pre-processing (with log-mel spectrogram inputs). Although these musically motivated architectures rely on strong assumptions about the nature of music, these still allow for expressive deep learning models. Without restricting too much the solution space, we aim to achieve more understandable expressive models that are more prone to generalize in the current context where large corpus of annotated audio are generally not available.

3.6 Publications, code and contributions

Out of this research, a series of conference articles were published:

- Jordi Pons, Thomas Lidy, and Xavier Serra. *"Experimenting with musically motivated convolutional neural networks"*, in 14th International Workshop on Content-Based Multimedia Indexing (CBMI2016).
  - This work was acknowledged with the best paper award.
  - Work done in collaboration with Thomas Lidy, who helped to conduct some preliminary experiments.
  - Code: https://github.com/jordipons/CBMI2016
- Jordi Pons, and Xavier Serra. *"Designing efficient architectures for modeling temporal features with convolutional neural networks"*, in 42nd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP2017).
  - Code: https://github.com/jordipons/ICASSP2017

- Jordi Pons, Olga Slizovskaia, Rong Gong, Emilia Gómez, and Xavier Serra. *"Timbre Analysis of Music Audio Signals with Convolutional Neural Networks"*, in 25th European Signal Processing Conference (EUSIPCO2017).

  - Code: https://github.com/jordipons/EUSIPCO2017
  - Code: https://github.com/Veleslavia/EUSIPCO2017
  - Code: https://github.com/ronggong/EUSIPCO2017
  - Work done in collaboration with Olga (who run the instrument classification experiments) and Rong (who run the singing-voice phoneme classification experiments).

Back when we started our research in 2015, it was not clear how deep learning would impact the music technology field. To experiment with deep learning and music audio, we decided to explore the musically motivated CNNs path as a way to adapt deep learning for the music audio case. To better understand which is the context and the scope of our contribution, see the feedback we got from an anonymous reviewer: *"Its contribution is that it goes beyond the general washing machine approach to deep learning (data in results out) to attempt to leverage/achieve insight on musical structure."*

In addition to the aforementioned research, together with Rong Gong[20], we also performed some additional experiments with musically motivated front-ends. In the context of his research project on automatically evaluating a cappella recordings for jingju (Beijing Opera) singing education, we published two articles:

[20] A former PhD student who was my colleague at the Music Technology Group.

- Jordi Pons, Rong Gong, and Xavier Serra. *"Score-informed syllable segmentation for a capella singing voice with convolutional neural networks"*, in 18th International Society for Music Information Retrieval Conference (ISMIR2017).

- Rong Gong, Jordi Pons, and Xavier Serra. *"Audio to score matching by combining phonetic and duration information"*, in 18th International Society for Music Information Retrieval Conference (ISMIR2017).

In short, in Pons et al. (2017a) and in Gong et al. (2017) we explored the usage of musically motivated CNNs as feature extractors that work together with probabilitstic models (e.g., with a Hidden Markov Model). In Pons et al. (2017a), we use the CNNs to estimate syllable onset detection functions, and in Gong et al. (2017) we use the CNNs to construct an acoustic model. In these works we show that musically motivated CNNs can be effective data-driven feature extractors — even if the training data is scarce.

After us, a series of very interesting follow-up publications (led by other researchers) kept investigating the characteristics and the applicability of musically motivated CNNs. For example, see the work by Chen and Wang (2017), Schreiber and Müller (2018), or Schreiber and Müller (2019). Furthermore, to see how the design strategy we proposed can be extended to other audio domains that are not music, see the work by Fonseca et al. (2018) — who employed similar ideas for acoustic scene classification.

Chapter 4

Non-trained CNNs for music and audio tagging

The computer vision literature shows that randomly weighted neural networks perform reasonably as feature extractors (Saxe et al., 2011; Rosenfeld and Tsotsos, 2018). Following this idea, we study how non-trained (randomly weighted) convolutional neural networks perform as feature extractors for (music) audio classification tasks. We use features extracted from the embeddings of deep architectures as input to a classifier — with the goal to compare classification accuracies when using different randomly weighted architectures. By following this methodology, we run a comprehensive evaluation of the current deep neural networks for audio classification, and provide evidence that the architectures alone are an important piece for resolving (music) audio problems using deep neural networks.

Along with this section we introduce a rather non-conventional technique to run a meta-evaluation of the main deep learning architectures for audio. By evaluating non-trained (randomly weighted)

CNNs, we run a meta-study at a low computational cost. To this end, we first perform a comprehensive state-of-the-art review and categorize the most used deep learning architectures for audio. In order to facilitate the discussion around these architectures, we divide the deep learning pipeline into two parts: front-end and back-end, see Figure 4.1. The front-end is the part that interacts with the input signal in order to map it into a latent-space, and the back-end predicts the output given the representation obtained by the front-end. Due to the nature of the proposed methodology (see below), the literature review we present focuses in introducing the main deep learning front-ends for audio classification: from the above introduced musically motivated CNNs (see Chapter 3), to the widely used computer vision architectures applied to spectrograms (Choi et al., 2016). A detailed review of the most used back-ends for audio classification is available in Chapter 5.

4.1 Motivation

Some intriguing properties of deep neural networks are periodically showing up in the scientific literature. Examples of those are: *(i)* perceptually non-relevant signal perturbations that dramatically affect the predictions of an image classifier (Goodfellow et al., 2014b; Szegedy et al., 2013); *(ii)* although there is no guarantee of converging to a global minima that might generalize, image classification models perform well with unseen data (Krizhevsky et al., 2012; He et al., 2015); or *(iii)* non-trained deep neural networks are able to perform reasonably well as image feature extractors (Saxe et al., 2011; Rosenfeld and Tsotsos, 2018). In this work, we exploit one of the above listed prop-

erties *(iii)* to evaluate how discriminative deep audio architectures are before training. Previous works already explored the idea of empirically studying the qualities of non-trained (randomly weighted) networks, but mainly in the computer vision field:

- Saxe et al. (2011) studied how discriminative are the architectures themselves by evaluating the classification performance of SVMs fed with features extracted from non-trained (random) CNNs. They showed that a surprising fraction of the performance in deep image classifiers can be attributed to the architecture alone. Therefore, the key to good performance lies not only on improving the learning algorithms but also in searching for the most suitable architectures. Further, they showed that the (classification) performance delivered by random CNN features is correlated with the results of their end-to-end trained counterparts — this result means, in practice, that one can bypass the time-consuming process of learning for evaluating a given architecture. We build on top of this result to evaluate current CNN architectures for audio classification.

- Rosenfeld and Tsotsos (2018) fixed most of the model's weights to be random, and only allowed a small portion of them to be learned. By following this methodology, they showed a small decrease in image classification performance when these models were compared to their fully trained counterparts. Further, the performance of their non fully-trained models can be summarized as follows:

$$DenseNet \gg ResNet > VGG \gg AlexNet \ [1]$$

[1] To know more about these computer vision architectures, see the original ref-

This result matches previous works reporting how these (fully trained) models perform (Huang et al., 2017; He et al., 2016; Simonyan and Zisserman, 2014), confirming the performance correlation between randomly weighted models and their trained counterparts found by Saxe et al. (2011)

- Adebayo et al. (2018) empirically assessed the local explanations of deep image classifiers to find that randomly weighted models produce explanations similar to those produced by models with learned weights. They conclude that the architectures introduce a strong prior which affects the learned (and not learned) representations.

- Ulyanov et al. (2018) also showed that the structure of a neural network (the non-trained architecture) is sufficient to capture useful features for the tasks of image denoising, super-resolution and inpainting. They think of any designed architecture as a *hand-crafted* model where prior information is embedded in the structure of the network itself. This way of thinking resonates with the rationale behind the family of audio models designed considering domain knowledge (see Section 4.2) — what denotes that in both audio and image fields it exists the interest of bringing together the end-to-end learning literature and previous research.

Few related works exist in the audio field, and every randomly weighted neural network we found in the audio literature was a mere baseline (Choi et al., 2017b; Kim et al., 2018; Arandjelovic and Zisserman,

erences (in order of appearance): Huang et al. (2017), He et al. (2016), Simonyan and Zisserman (2014), and Krizhevsky et al. (2012)

2017). Inspired by previous computer vision works, we study which audio architectures work the best via evaluating how non-trained CNNs perform as feature extractors. To this end, we use the CNNs' embeddings to construct feature vectors for a classifier — with the goal to compare classification performances when different randomly weighted architectures are used to extract features.

Extreme learning machines (ELMs) (Schmidt et al., 1992; Pao et al., 1994; Huang et al., 2006), and echo state networks (ESNs) (Jaeger, 2001) are also closely related to our work. In short, ELMs are classification/regression models[2] that are based on a single-layer MLP with random weights. ELMs work as follows: first, they randomly project the input into a latent space; and then, learn how to predict the output via a least-square fit. More formally, one aims to predict:

$$\hat{Y} = W_2 \, \sigma(W_1 X), \tag{4.1}$$

where W_1 is the (randomly weighted) matrix of input-to-hidden-layer weights, σ is the non-linearity, W_2 is the matrix of hidden-to-output-layer weights, and X represents the input. The training algorithm is as follows: *1)* set W_1 with random values; *2)* estimate W_2 via a least-squares fit:

$$W_2 = \sigma(W_1 X)^{+} Y, \tag{4.2}$$

where $^{+}$ denotes the Moore-Penrose inverse. Since no iterative process is required for learning the weights, training is faster than stochastic gradient descent (Huang et al., 2006). Provided that we process audio signals with randomly weighted CNNs, ELM-based classifiers are a natural choice for our study — so that all the pipeline (except

[2]Support Vector Machines are also classification/regression models.

the last layer) is based on random projections that are only constrained by the structure of the neural network. Although ELMs are not widely used by the audio community, they have been used for speech emotion recognition (Han et al., 2014; Kaya and Salah, 2016), or for music audio classification (Scardapane et al., 2013; Loh and Emmanuel, 2006; Khoo et al., 2012). ESNs differ from ELMs in that their random projections use recurrent connections. Given that the audio models we aim to study are not recurrent, we leave for future work using ESNs — see Scardapane and Uncini (2017) or Holzmann (2009) for audio applications of ESNs.

4.2 Literature review: CNN front-ends for audio

Along this chapter we evaluate the most used deep learning architectures for (music) audio classification. In order to facilitate the discussion around these architectures, we divide the deep learning pipeline into two parts: front-end and back-end.

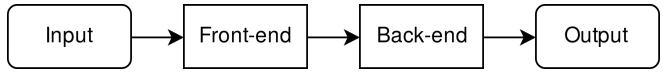


Figure 4.1: The deep learning pipeline.

The front-end is the part that interacts with the input signal in order to map it into a latent-space, and the back-end predicts the output given the representation obtained by the front-end. Note that one can interpret the front-end as a "feature extractor" and the back-end as a "classifier". Given that we compare how several non-trained (random) CNNs perform as feature extractors, and we will

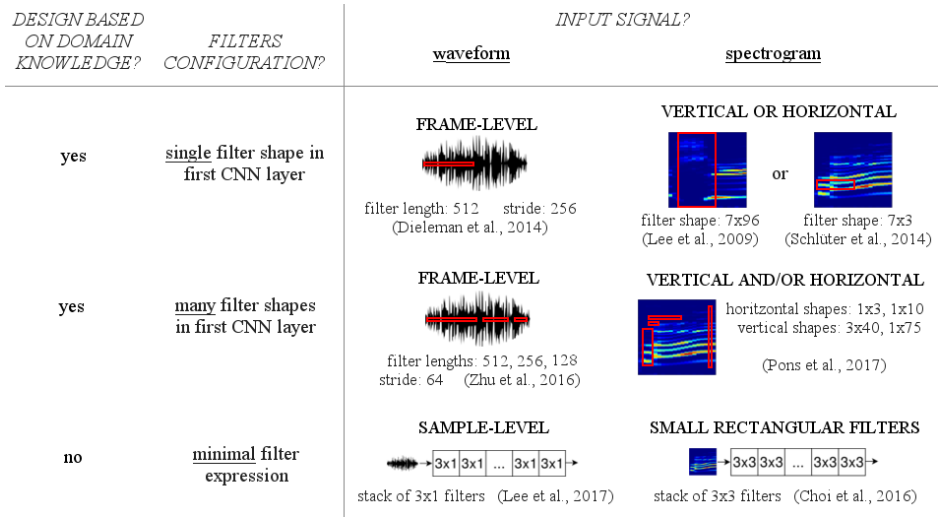


Figure 4.2: CNN front-ends for audio classification tasks — with examples of possible configurations for every paradigm.

use out-of-the-box classifiers to predict the classes: this literature review focuses in introducing the main deep learning front-ends for audio classification.

Front-ends — These are generally conformed by CNNs (Dieleman and Schrauwen, 2014; Choi et al., 2016; Zhu et al., 2016), since these can encode efficient representations by sharing weights[3] along the signal. Figure 4.2 depicts six different CNN front-end paradigms, which can be divided into two groups depending on the used input signal: waveforms (Dieleman and Schrauwen, 2014; Zhu et al., 2016; Lee et al., 2018) or spectrograms (Choi et al., 2016; Pons and Serra, 2017; Pons et al., 2017b). Further, the design of the filters can be either based on domain knowledge or not. For example, one leverages domain knowledge when the *frame-level single-*

[3]Which constitute the (eventually learnt) feature representations.

shape[4] front-end for waveforms is designed so that the length of the filter is set to be the same as the window length in a STFT (Dieleman and Schrauwen, 2014). Or for spectrogram front-ends, one laverages domain knowledge when using *vertical* filters to learn spectral representations (Lee et al., 2009) *or horizontal* filters to learn longer temporal cues (Schlüter and Böck, 2014). Generally, a single filter shape is used in the first CNN layer (Dieleman and Schrauwen, 2014; Choi et al., 2016; Lee et al., 2009; Schlüter and Böck, 2014), but some recent work reported performance gains when using several filter shapes in the first layer (Zhu et al., 2016; Pons and Serra, 2017; Pons et al., 2017b; Phan et al., 2016; Pons et al., 2016b; Chen and Wang, 2017). Using many filters promotes a more rich feature extraction in the first layer, and facilitates leveraging domain knowledge for designing the filters' shape. For example: a *frame-level many-shapes* front-end for waveforms can be motivated from a multi-resolution time-frequency transform[5] perspective — with window sizes varying inversely with frequency (Zhu et al., 2016). Or since it is known that some patterns in spectrograms are occurring at different time-frequency scales, one can intuitively incorporate *many vertical and/or horizontal* filters to efficiently capture those in a spectrogram front-end (Pons and Serra, 2017; Pons et al., 2017b). As seen, using domain knowledge when designing the models allows to naturally connect the deep learning literature with previous relevant signal processing work. On the other hand, when domain knowledge is not used, it is common to employ a deep stack of small filters, e.g.: 3×1 in the *sample-level* front-end used for waveforms (Lee et al., 2018; van den Oord et al., 2016), or 3×3 in the *small-rectangular filters* front-end used for spectro-

[4]Italicized names correspond to the front-end types in Figure 4.2.

[5]The Constant-Q Transform (Brown, 1991) is an example of such transform.

grams (Choi et al., 2016). These models based on a deep stack of small filters make minimal assumptions over the local stationarities of the signal, so that any structure can be learnt via hierarchically combining small-context representations.

4.3 Methodology

Our goal is to study which CNN front-ends work best via evaluating how non-trained models perform as feature extractors. Our methodology is based on the traditional pipeline of *features extraction + classifier*. We use the embeddings of non-trained (random) CNNs as features: for every audio clip, we compute the average of each feature map (in every layer) and concatenate these values to construct a feature vector (Choi et al., 2017b). The baseline feature vector is constructed from 20 MFCCs, their Δs and $\Delta\Delta$s. We compute their mean and standard deviation through time, and the resulting feature vector is of size 120. We set the standard MFCCs + SVM setup as baseline. To allow a fair comparison with the baseline, CNN models have ≈ 120 feature maps — so that the resulting feature vectors have a similar size as the MFCCs vector. Further, we evaluate an alternative configuration with more feature maps (≈ 3500) to show the potential of this approach. The model's description omit the number of filters per layer for simplicity — full implementation details are accessible online, see Section 5.7 for the links.

Features: randomly weighted CNNs

Except for the *VGG* model that uses ELUs as non-linearities (Choi et al., 2016; Clevert et al., 2016), the rest use ReLUs (He et al., 2015). We do not use batch normalization, see discussion in section 4.4.

We use waveforms and spectrograms as input to our CNNs:

Waveform inputs — are of ≈ 29sec (350,000 samples at 12kHz), and the following architectures are under study:

- *Sample-level:* is based on a stack of 7 blocks that are composed by a 1D-CNN layer (filter length: 3, stride: 1) and a max-pool layer (size: 3, stride: 3), with the exception of the input block which has no max-pooling and its 1D-CNN layer has a stride of 3 (Lee et al., 2018). Averages to construct the feature vector are computed after every pooling layer, except for the first layer that are computed after the CNN.

- *Frame-level many-shapes:* consists of a 1D-CNN layer with five filter lengths: 512, 256, 128, 64, 32 (Zhu et al., 2016). Every filter's stride is of 32 and we use *same* padding — to easily concatenate feature maps of the same size. Note that out of this single 1D-CNN layer, five feature maps (resulting of the different filter length convolutions) are concatenated. For that reason, every feature map needs to have the same (temporal) size. Since this model is single-layered and it might be in clear disadvantage when compared to the *sample-level* CNN, we increase its depth via adding three more 1D-CNN layers (length: 7, stride: 1) — where the last two layers have residual connections, and the penultimate layer's feature map is down-sampled by two (MP x2), see Figure 4.3. Averages to construct the feature vector are calculated for each feature map after every 1D-CNN layer.

- *Frame-level:* consists of a 1D-CNN layer with a filter of length 512 (Dieleman and Schrauwen, 2014). Stride is set to be 32 to allow a fair comparison with *frame-level many-shapes*. As in *frame-level many-shapes*, we increase the depth of the model via adding three more 1D-CNN layers (as in Figure 4.3). Averages to construct the feature vector are calculated for each feature map after every 1D-CNN layer.

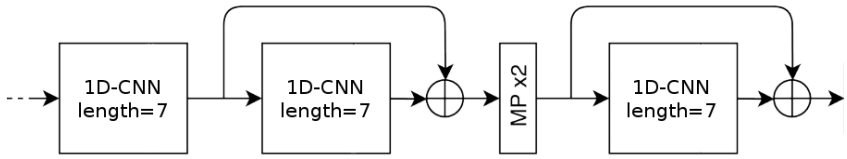


Figure 4.3: Additional layers for the *frame-level* & *frame-level many-shapes* architectures, similar as in Dieleman and Schrauwen (2014) and in Pons et al. (2018) — where MP stands for max pooling.

Spectrogram inputs — are set to be log-mel spectrograms (spectrograms size: 1376×96[6], being ≈ 29sec of signal). Differently from waveform models, spectrogram architectures use no additional layers to deepen single-layered CNNs because these already deliver a reasonable classification performance. Unless we state the contrary, every CNN layer used for processing spectrograms is set to have a stride of 1. As for the *frame-level many-shapes* model, we use *same* padding when many filter shapes are used in the same layer. The following spectrogram models are studied:

- *7×96*: consists of a single 1D-CNN layer with filters of length 7 that convolve through the time axis (Dieleman and Schrauwen, 2014). As a result: CNN filters are vertical and of shape 7×96. Therefore, these filters can encode spectral (timbral) representations. Averages to construct the feature vector are calculated for each feature map after the 1D-CNN layer.

- *7×86*: consists of a single 2D-CNN layer with vertical filters of shape 7×86 (Pons et al., 2017b, 2016b). Since its vertical shape is smaller than the input (86<96), filters can also convolve through the frequency axis — what can be seen as "pitch shifting" the filter. Consequently, 7×86 filters can encode pitch-invariant timbral

[6]STFT parameters: *window_size = 512*, *hop_size=256*, and *fs=12kHz*.

representations (Pons et al., 2017b, 2016b). Further, since the resulting activations can carry pitch-related information, we max-pool the frequency axis to get pitch-invariant features (max-pool shape: 1×11). Averages to construct the feature vector are calculated for each feature map after the max-pool layer.

- *Timbral*: consists of a single 2D-CNN layer with many vertical filters of shapes: 7×86, 3×86, 1×86, 7×38, 3×38, 1×38, see Figure 4.4 (top) (Pons et al., 2017b). These filters can also convolve through the frequency axis and therefore, these can encode pitch-invariant representations. Several filter shapes are used to efficiently capture different timbrically relevant time-frequency patterns, e.g.: kick-drums (can be captured with 7×38 filters representing subband information for a short period of time), or string ensemble instruments (can be captured with 1×86 filters representing spectral patterns spread in the frequency axis). Further, since the resulting activations can carry pitch-related information, we max-pool the frequency axis to get pitch-invariant features (max-pool shapes: 1×11 or 1×59). Averages to construct the feature vector are calculated for each feature map after the max-pool layer.

- *Temporal*: several 1D-CNN filters (of lengths: 165, 128, 64, 32) operate over an energy envelope obtained via mean-pooling the frequency-axis of the spectrogram, see Figure 4.4 (bottom). By computing the energy envelope in that way, we are considering high and low frequencies together while minimizing the computations of the model. Observe that this single-layered 1D-CNN is not operating over a 2D spectrogram, but over a 1D energy envelope — therefore no vertical convolutions are performed, only 1D (temporal) convolutions are computed. As seen, domain knowledge can also provide

guidance to (effectively) minimize the computations of the model. Averages to construct the feature vector are calculated for each feature map after the CNN layer.

- *Timbral+temporal*: combines both *timbral* and *temporal* CNNs in a single (but wide) layer, see Figure 4.4 (Pons et al., 2018). Averages to construct the feature vector are calculated in the same way as for *timbral* and *temporal* architectures.

- *VGG*: is a computer vision model based on a stack of 5 blocks combining 2D-CNN layers (with small rectangular filters of 3×3) and max-pooling (of shapes: 4×2, 4×3, 5×2, 4×2, 4×4, respectively) (Choi et al., 2016). Averages to construct the feature vector are computed after every pooling layer.

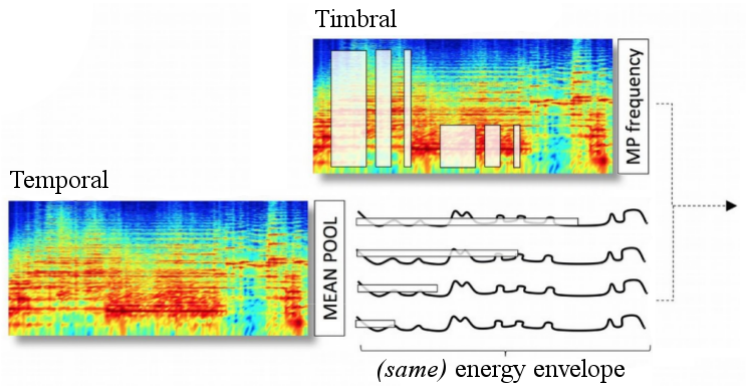


Figure 4.4: *Timbral+temporal* architecture. MP: max-pool.

As seen, studied architectures are representative of the audio classification state-of-the-art (introduced in Section 4.2). For further details about the models under study: the code is online[7], and a graphical conceptualization of the models is in Figures 4.2, 4.3 and 4.4.

[7]In Section 5.7 (Contributions) we attach the link to the source code.

Classifiers: SVM and ELM

We study how several feature vectors (computed considering different CNNs) perform for a given set of classifiers: SVMs and ELMs. We discarded the use of other classifiers since their performance was not competitive when compared to those. SVMs and ELMs are hyper-parameter sensitive, for that reason we perform a grid search:

- *SVM* hyper-parameters: we consider both *linear* and *rbf* kernels. For the *rbf* kernel, we set γ to: 2^{-3}, 2^{-5}, 2^{-7}, 2^{-9}, 2^{-11}, 2^{-13}, $\#features^{-1}$; and for every kernel configuration, we try several C's (penalty parameter): 0.1, 2, 8, 32.

- *ELM*'s main hyper-parameter is the number of hidden units: 100, 250, 500, 1200, 1800, 2500. We use ReLUs as non-linearity.

Datasets: music and acoustic events

- *GTZAN* fault-filtered version (Tzanetakis and Cook, 2002; Kereliuk et al., 2015). Training songs: 443, validation songs: 197, and test songs: 290 — divided in 10 classes. We use this dataset to study how randomly weighted CNNs perform for music genre classification.

- *Extended Ballroom* (Marchand and Peeters, 2016; Cano et al., 2006). 4,180 songs are divided in 13 classes. 10 stratified folds are randomly generated for cross-validation. We use this dataset to study how randomly weighted CNNs classify rhythm/tempo classes.

- *Urban Sound 8K* (Salamon et al., 2014a). 8732 acoustic events are divided in 10 classes. 10 folds are already defined by the dataset authors for cross-validation. Since urban sounds are shorter than 4 seconds and our models accept $\approx$ 29sec inputs, the signal is repeated to create inputs of the same length. We use this dataset to study how randomly weighted CNNs perform to classify (non-music) sounds.

4.4 Reproducing former results to discuss our method

Choi et al. (2017b) used random CNN features as baseline for their work, and found that (in most cases) these random CNN features perform better than MFCCs. Motivated by this result, we pursue this idea for studying how different deep architectures perform when resolving audio problems. To start, we first reproduce one of their experiments using random CNNs — under the same conditions[8]: the GTZAN dataset is split in 10 stratified folds used for cross-validation[9], a VGG architecture with batch normalization is employed, and the classifier is an SVM. We found that results can vary depending on the batch size if, when computing the feature vectors, layers are normalized with the statistics of current batch (batch normalization). For example: if audio-features of the same genre are batch-normalized by the same factor, one can create an artificial *genre cue* that might affect the results. One can observe this phenomena in Figure 4.5, where the best results are achieved when all songs of the same genre fill a full batch (batch size of 100).[10] We also observe that small batch sizes are harming the model's performance — see in Figure 5 when batch sizes are set to 1 and 10. And finally, when batch normalization is not used, no matter what batch size we use that the results remain the same — *ANOVA* test with H_0 being that averages are equal (*p-value=0.491*). Since it is not desirable that

[8] https://github.com/keunwoochoi/transfer_learning_music *(more information is available in issue #2)*

[9] Our work does not use this split, we use the fault-filtered version.

[10] The GTZAN has 10 genres with 100 audios each, one can fill batches of 100 with audios of the same genre via sorting the data by genres.

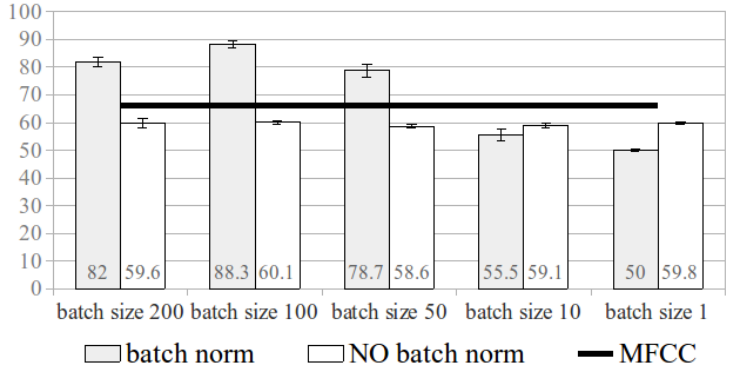


Figure 4.5: Random CNN features behavior when using (or not) batch normalization. *Dataset*: GTZAN, 10-fold cross-validation. *Performance metric (%)*: average accuracies (and standard deviations) across 3 runs. *Classifier*: SVM.

performance depends on the batch size, and that the feature vector of an audio depends on other audios (that are present in the batch): we decided not to use batch normalization.

4.5 Experimental results

The results below show average accuracies across 3 runs for every feature type (listed on the right side of each figure with the length of the feature vector in parenthesis). We use a t-test to reveal which models are performing the best — H_0 being that averages are equal.

4.5.1 GTZAN: music genre recognition

The *sample-level* waveform model always performs better than *frame-level many-shapes* (t-test: p-value$\ll$0.05). The two best performing spectrogram-based models are: *timbral+temporal* and *VGG* —

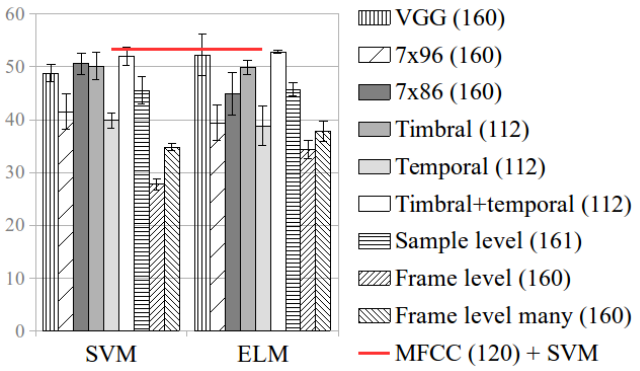


Figure 4.6: Accuracy (%) results for the GTZAN dataset with random CNN feature vectors of length ≈ 120.

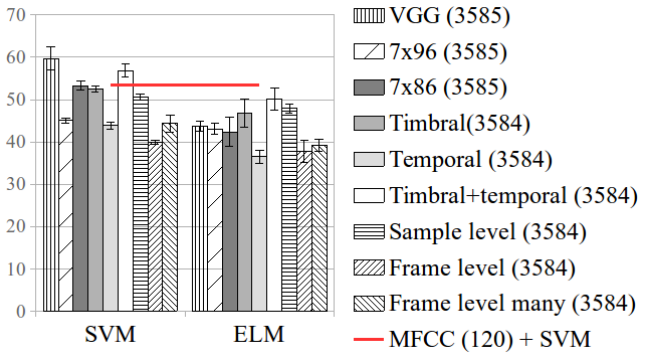


Figure 4.7: Accuracy (%) results for the GTZAN dataset with random CNN feature vectors of length ≈ 3500.

with a remarkable performance of the *timbral* model alone. The *timbral+temporal* CNN performs better than *VGG* when using the ELM (≈3500) classifier (t-test: p-value=0.017); but in other cases, both models perform equivalently (t-test: p-value>0.05). Moreover, the *7x86* model performs better than *7x96* when using SVMs (t-test: p-value<0.05), but when using ELMs: *7x96* and *7x86* perform equivalently (t-test: p-value≫0.05). The best *VGG* and *timbral+temporal* models achieve the following (average) accuracies: 59.65% and 56.89%, respectively — both with an SVM (≈3500) classifier. Both models outperform the MFCCs baseline: 53.44% (t-test: p-value<0.05), but these random CNNs perform worse than a CNN pre-trained with the Million Song Dataset: 82.1% (Lee et al., 2018). Finally, note that although *timbral* and *timbral+temporal* models are single-layered, these are able to achieve remarkable performances — showing that single-layered spectrogram front-ends, attending to musically relevant contexts, can do a reasonable job without paying the computational cost of going deep (Pons et al., 2017b, 2016b). Thus meaning, e.g., that the saved capacity can now be employed by the back-end to learn additional representations.

4.5.2 Ballroom: rhythm/tempo classification

The *sample-level* waveform model always performs better than *frame-level many-shapes* (t-test: p-value≪0.05). The two best performing spectrogram-based models are: *temporal* and *timbral+temporal*, but the *temporal* model performs better than *timbral+temporal* in all cases (t-test: p-value≪0.05) — denoting that spectral cues can be a confounding factor for this dataset. Moreover, the *7x86* model performs better than *7x96* in all cases (t-test: p-value<0.05). The

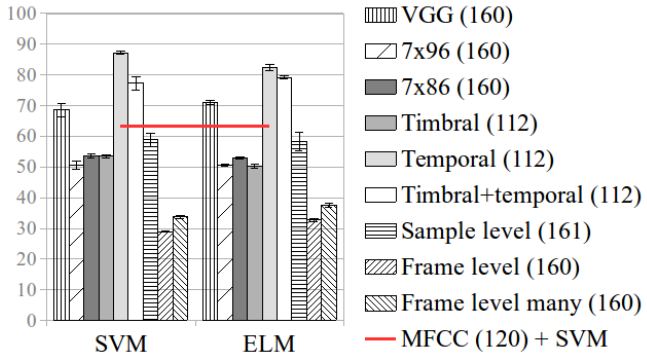


Figure 4.8: Accuracy (%) results for the Extended Ballroom dataset with random CNN feature vectors of length ≈ 120.

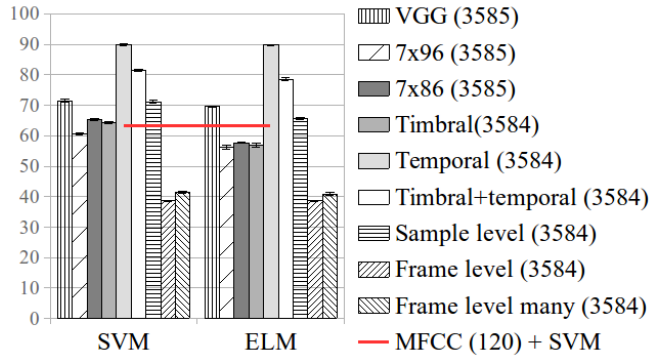


Figure 4.9: Accuracy results for the Extended Ballroom dataset with random CNN feature vectors of length ≈ 3500.

best (average) accuracy score is obtained using *temporal* models and SVMs (≈3500): 89.82%. Note that the *temporal* model clearly outperforms the MFCCs baseline: 63.25% (t-test: p-value$\ll$0.05) and, interestingly, it performs slightly worse than a trained CNN: 93.7% (Jeong et al., 2017). This result confirms that the architectures (alone) introduce a strong prior which can significantly affect the performance of an audio model. Thus meaning that, besides learning, designing effective architectures might be key for resolving (music) audio tasks with deep learning. In line with that, note that the *temporal* architecture is designed considering musical domain knowledge. In this case: how tempo and rhythm are expressed in spectrograms. Hence, its good performance also validates the design strategy of using musically motivated architectures as a way to intuitively navigate through the network parameters space (Pons et al., 2016b, 2017b).

4.5.3 Urban Sound 8K: acoustic event detection

For these experiments we do not use the *temporal* model (with 1D-CNNs of length 165, 128, 64, 32). Instead, we study the *timbral+time* model — where *time* follows the same design as *temporal* but with filters of length: 64, 32, 16, 8. This change is motivated by the fact that temporal cues in (natural) sounds are shorter and less important than temporal cues in music (i.e., rhythm or tempo).

The *sample-level* waveform model always performs better than *frame-level many-shapes* (t-test: p-value$\ll$0.05). The two best performing spectrogram-based models are: *VGG* and *timbral+time* — but *VGG* performs better than *timbral+time* in all cases (t-test: p-value$\ll$0.05). Also, the *7x86* model performs better than *7x96* in all cases (t-test: p-value$<$0.075). The best (average) accuracy score is obtained us-

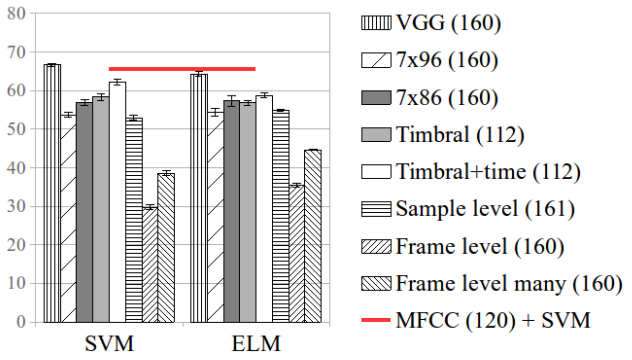


Figure 4.10: Accuracy (%) results for the Urban Sound 8k dataset with random CNN feature vectors of length ≈ 120.

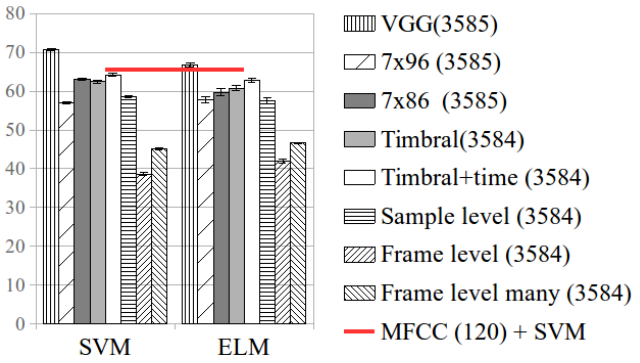


Figure 4.11: Accuracy (%) results for the Urban Sound 8k dataset with random CNN feature vectors of length ≈ 3500.

ing *VGG* and SVMs (≈3500): 70.74% — outperforming the MFCCs baseline: 65.49% (t-test: p-value<0.05), and performing slightly worse than a trained CNN: 73%[11] (Salamon and Bello, 2017). Finally, note that *VGG*s achieved remarkable results when recognizing genres and detecting acoustic events — tasks where timbre is an important cue. As a result: one could argue that *VGG*s are good at representing spectral features. Hence, these might be of utility for tasks where spectral cues are relevant.

4.6 Summary and conclusions

This study builds on top of prior works showing that the (classification) performance delivered by random CNN features is correlated with the results of their end-to-end trained counterparts (Saxe et al., 2011; Rosenfeld and Tsotsos, 2018). We use this property to run a comprehensive evaluation of the main deep architectures for (music) audio. Our method is as follows: first, we extract a feature vector from the embeddings of a randomly weighted CNN; and then, we input these features to a classifier — which can be an SVM or an ELM. Our goal is to compare the obtained classification accuracies when using different CNN architectures. The results we obtain are far from *random*, since: *(i)* randomly weighted CNNs are (in some cases) close to match the accuracies obtained by trained CNNs; and *(ii)* these are able to outperform MFCCs. This result denotes that the architectures alone are an important piece of the deep learning solution and therefore, searching for efficient architectures capable to encode the specificities of (music) audio signals

[11] The same CNN achieves 79% when trained with data augmentation.

might help advancing the state of our field. In line with that, we have shown that (musical) priors embedded in the structure of the model can facilitate capturing useful (temporal) cues for classifying rhythm/tempo classes. Besides, we show that for waveform front-ends: *sample-level* $\gg$ *frame-level many-shapes* $>$ *frame-level*, as noted in the (trained) literature (Lee et al., 2018; Zhu et al., 2016; van den Oord et al., 2016). The differential aspect of the *sample-level* front-end is that its representational power is constructed via hierarchically combining small-context representations, not by exploiting prior knowledge about waveforms. Further, we show that for spectrogram front-ends: *7x96*$<$*7x86*, as shown in prior (trained) works (Pons et al., 2016b; Oramas et al., 2017). By allowing the filters to convolve through the frequency axis, the architecture itself facilitates capturing pitch-invariant timbral representations. Finally: *timbral* (*+temporal/time*) and *VGG* spectrogram front-ends achieve remarkable results for tasks where timbre is important — as previously noted in the (trained) literature (Pons et al., 2017b; Choi et al., 2016). Their respective advantages being that: *(i) timbral* (*+temporal/time*) architectures are single-layered front-ends which explicitly capture acoustically relevant receptive fields — which can be known via exploiting prior knowledge about the task; and *(ii) VGG* front-ends require no prior domain knowledge about the task for its design. Although our main conclusions are backed by additional results in the (trained) literature, we leave for future work consolidating those via doing a similar study considering trained models.

4.7 Publications, code and contributions

Out of this research, a conference article was published:

- Jordi Pons, and Xavier Serra. *"Randomly weighted CNNs for (music) audio classification"*, in 44th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP2019).
  - Code: https://github.com/jordipons/elmarc

To the best of our knowledge, this is the first comprehensive evaluation of randomly weighted CNNs for (music) audio classification. Thus providing a first meta-evaluation of the most used deep learning architectures for audio. In addition, we found remarkable that non-trained CNNs can extract discriminative audio features at all. Particularly, we want to highlight the results achieved by the random CNNs processing waveforms, that are just relying on random projections to extract features from a high-dimensional signal.

Finally, we also want to highlight two minor contributions that are closely related to the research we present along that chapter:

- Alongside with our literature review, we provide a categorization of the main CNN architectures for audio (Figure 4.2) that has proven to be very useful for didactic porupuses.[12]
- Extreme Learning Machines (ELMs) are not widely known by music and audio researchers. We introduce ELMs to the audio research community, and show that these can perform competently. In line with that, we also prepared a blog-post to help to disseminate ELMs among audio researchers.[13]

[12] See these slides: www.jordipons.me/media/UPC-2018.pdf
[13] www.jordipons.me/extreme-learning-machines-for-audio

Chapter 5

Music tagging at scale

As we noted in our introductory review (Chapters 1 and 2), the lack of data tends to limit the outcomes of deep learning research — particularly when dealing with end-to-end learning stacks processing raw data such as waveforms. In this study, we collaborated with Pandora Radio for accessing to 1.2M annotated music tracks for training our end-to-end models. This large amount of data allowed us to unrestrictedly explore two different design paradigms for music auto-tagging: *(i)* assumption-free models — using waveforms as input with very small convolutional filters; and *(ii)* models that heavily rely on domain knowledge — log-mel spectrograms with a CNN designed to learn timbral and temporal features (musically motivated CNNs). Our work focuses on studying how these two types of deep architectures perform when datasets of variable size are available for training: the MagnaTagATune (25k songs), the Million Song Dataset (240k songs), and a private dataset of 1.2M songs. Our experiments suggest that music domain assumptions are relevant when not enough training data are available, thus showing how waveform-based models can outperform spectrogram-based ones in large-scale data scenarios.

5.1 Introduction

One fundamental goal in music informatics research is to automatically structure large music collections. The music audio tagging task consists of automatically estimating the musical attributes of a song — including: moods, language of the lyrics, year of composition, genres, instruments, harmony, or rhythmic traits. Thus, tag estimates may be useful to define a semantic space that can be advantageous for automatically organizing musical libraries.

Many approaches have been considered for this task. Some of those have been based on the *feature extraction + model* pipeline (Bayle et al., 2017; Sordo et al., 2007; Prockup et al., 2015), while recent publications are showing promising results using deep architectures (Dieleman and Schrauwen, 2014; Choi et al., 2016; Pons et al., 2017b; Lee et al., 2018). In this work we confirm this trend by studying how two deep architectures conceived considering opposite design strategies (using domain knowledge or not) perform for several datasets — with one of the datasets being of an unprecedented size: 1.2M songs. Provided that a sizable amount of data is available for that study, we investigate the learning capabilities of these two architectures. Specifically, we investigate whether the architectures based on domain knowledge overly constrain the solution space for cases where large training data are available. In essence, we study if certain architectural choices (e.g., using log-mel spectrograms as input) can limit the model's capabilities to learn from data. The main contribution of this work is to show that little to no model assumptions are required for music auto-tagging when operating with large amounts of data.

5.2 Literature review: deep learning architectures

In order to facilitate the discussion around the current audio architectures, we divide deep learning pipeline into two parts: front-end and back-end (see Figure 5.1). The front-end is the part of the model that interacts with the input signal in order to map it into a latent-space, and the back-end predicts the output given the representation obtained by the front-end. In the following, we present the main front- and back-ends we identified in the literature.

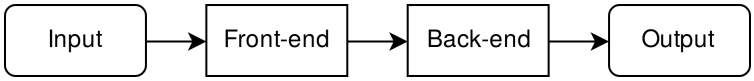


Figure 5.1: The deep learning pipeline.

Front-ends[1] — These are generally comprised of CNNs (Dieleman and Schrauwen, 2014; Choi et al., 2016; Zhu et al., 2016; Pons and Serra, 2017; Pons et al., 2017b), since these can learn efficient representations by sharing weights[2] along the signal. Front-ends can be divided into two groups depending on the used input signal: waveforms (Dieleman and Schrauwen, 2014; Zhu et al., 2016; Lee et al., 2018) or spectrograms (Choi et al., 2016; Pons and Serra, 2017; Pons et al., 2017b). Further, the design of the filters can be either based on domain knowledge or not. For example, one leverages domain knowl-

[1]This overview is closely related to the front-ends review we introduced in Section 4, that is also presented in Pons and Serra (2019). Both depart from the same analysis, and from the same categorization (depicted in Figure 4.2).

[2]Which determine the learned feature representations.

edge when a front-end for waveforms is designed so that the length of the filter is set to be as the window length of a STFT (Dieleman and Schrauwen, 2014). Or for a spectrogram front-end, it is used vertical filters to learn timbral representations (Lee et al., 2009) or horizontal filters to learn longer temporal cues (Schlüter and Böck, 2014). Generally, a single filter shape is used in the first CNN layer (Dieleman and Schrauwen, 2014; Choi et al., 2016; Lee et al., 2009; Schlüter and Böck, 2014), but some recent works report performance gains when using several filter shapes in the first layer (Zhu et al., 2016; Pons and Serra, 2017; Pons et al., 2017b; Phan et al., 2016; Pons et al., 2016b). Using many filters promotes a richer feature extraction in the first layer, and facilitates leveraging domain knowledge for designing the filters' shape. For example: a waveform front-end using many long filters (of different lengths) can be motivated from the perspective of a multi-resolution time-frequency transform[3] (Zhu et al., 2016); or since it is known that some patterns in spectrograms are occurring at different time-frequency scales, one can intuitively incorporate many (different) vertical and/or horizontal filters in a spectrogram front-end (Pons and Serra, 2017; Pons et al., 2017b, 2016b; Phan et al., 2016). As seen, using domain knowledge during the design process allows us to naturally connect the deep learning literature with previous signal processing work. On the other hand, when domain knowledge is not used, it is common to employ a deep stack of small filters, e.g.: 3×1 as in the sample-level front-end used for waveforms (Lee et al., 2018), or 3×3 filters used for spectrograms (Choi et al., 2016). These models based on small filters make minimal assumptions over the local stationarities of the signal, so that any structure can be learned

[3]The Constant-Q Transform (Brown, 1991) is an example of such transform.

via hierarchically combining small-context representations. These architectures with small filters are flexible models able to potentially learn any structure given enough depth and data.

Back-ends — Among the different back-ends used in the audio literature, we identified two main groups: *(i)* fixed-length input back-end, and *(ii)* variable-length input back-end. The generally convolutional nature of the front-end allows it to process different input lengths. Therefore, the back-end unit can adapt a variable-length feature map to a fix-sized output. The former group of models (*i*) assume that the input will be kept constant – examples of those are front-ends based on feed-forward neural-networks or fully-convolutional stacks (Dieleman and Schrauwen, 2014; Choi et al., 2016). The second group (*ii*) can deal with different input-lengths since the model is flexible in at least one of its input dimensions. Examples of those are back-ends using temporal-aggregation strategies such as max-pooling, average-pooling, attention models or recurrent neural networks (Raffel, 2016). Given that songs are generally of different lengths, these types of back-ends capable to deal with variable-length inputs are ideal candidates for music processing. However, despite the different-length nature of music, many works employ fixed-length input back-ends (group *i*) since these architectures tend to be simpler and perform well (Choi et al., 2016; Pons et al., 2017b; Dieleman and Schrauwen, 2014).

5.3 Datasets under study

We study how two different deep architectures for music auto-tagging perform for three music collections of different sizes:

(i) The **MagnaTagATune** (MTT) dataset is of ≈ 26k music audio clips of 30s (Law et al., 2009). Predicting the top-50 tags of this dataset is a popular benchmark for music auto-tagging.

(ii) Although the **Million Song Dataset** (MSD) name indicates that 1M songs are available (Bertin-Mahieux et al., 2011), audio files with proper tag annotations (top-50 tags) are only available for ≈ 240k previews of 30s. This dataset constitutes the biggest public dataset available for music auto-tagging, making these data highly appropriate for benchmarking.

(iii) A private dataset consisting of 1M songs for training, 100k for validation, and 100k for test[4] is available for this study. The **1.2M-songs** dataset has 139 track-level human-expert annotations that can be summarized as follows:

- *Meter tags* denote different sorts of musical meters (e.g., triple-meter, cut-time, compound-duple, odd).
- *Rhythmic feel tags* denote rhythmic interpretation (e.g., swing, shuffle, back-beat strength) and elements of rhythmic perception (*e.g.*, syncopation, danceability).
- *Harmonic tags*: major, minor, chromatic, etc.
- *Mood tags* express the sentiment of a music audio clip (e.g., if the music is angry, sad, joyful).
- *Vocal tags* denote the presence of vocals and timbral characteristics of it (e.g., male, female, vocal grittiness).

[4]Test and validation sets are kept the same throughout the experiments for a fair evaluation. All used partitions are stratified and artist-filtered.

- *Instrumentation tags* denote the presence of instruments (e.g., piano) and their timbre (e.g., guitar distortion).
- *Sonority tags* detail production techniques (e.g., studio, live) and overall sound (e.g., acoustic, synthesized).
- *Subgenre tags*: jazz (e.g., cool, fusion, hard bop), rock (e.g., light, hard, punk), rap (e.g., east coast, old school), world music (e.g., cajun, indian), classical music (e.g., baroque period, classical period), etc.

Other large (music) audio datasets exist: the Free Music Archive by Defferrard et al. (2017) (FMA: $\approx$ 106k songs), and Audioset by Gemmeke et al. (2017) ($\approx$ 2.1M audios). Since previous works mainly used the MTT and MSD (Choi et al., 2016; Lee et al., 2018), we employ these datasets to assess the studied models with public data. Despite our interest in using FMA, for brevity, we restrict our study to 3 datasets that already cover a wide range of different sizes. Finally, Audioset is not used since most of its content is not music.

5.4 Architectures under study

After an initial exploration of the different architectures we identified in the literature, we select two models based on opposite design paradigms: one for processing waveforms, with a design that does minimal assumptions over the task at hand; and another for spectrograms, with a design that heavily relies on musical domain knowledge. Our goal is to compare these two models for providing insights in whether domain knowledge is required (or not) for designing deep learning models. This section provides discussion around our architectural choices and introduces the *basic* configuration setup.

The waveform model was selected after observing that the sample-level front-end (using a deep stack of 3×1 filters) was remarkably superior to the other waveform-based front ends — as shown in the original paper Lee et al. (2018). This result is particularly compelling because this front-end does not rely on domain-knowledge for its design. Note that raw waveforms are fed to the model without any pre-processing, and the small filters considered for its design make no strong assumptions over which are the most informative local stationarities in waveforms. Therefore, the sample-level can be seen as a problem agnostic front-end that has the potential to learn any audio task provided that enough depth and data are available. Given that a large amount data is available for this study, the sample-level front-end is of particular interest due to its strong learning potential: its solution space is not constrained by severe architectural choices relying on domain knowledge.

On the other hand, when experimenting with spectrogram front-ends, we found domain knowledge intuitions to be valid guides for designing deep architectures. For example, front-ends based on *(i)* many vertical and horizontal filters in the first layer were consistently superior to front-ends based on *(ii)* a single vertical filter — as shown in recent publications (Chen and Wang, 2017; Pons and Serra, 2017; Pons et al., 2016b; Phan et al., 2016). Note that the former front-ends *(i)* can learn spectral and (long) temporal representations already in the first layer, which are known to be important musical cues; while the latter *(ii)* can only learn spectral representations. Moreover, we observed that front-ends based on a deep stack of 3×3 filters were achieving equivalent performances to the former front-end *(i)* when input segments were shorter than 10s. We also found this behavior

in previous works (Pons et al., 2017b). But when considering longer inputs (which yielded better performance), the computational price of this deeper model increases: longer inputs implies having larger feature maps in every layer and therefore, more GPU memory consumption. For that reason, we refrained from using a deep stack of 3×3 filters as a front-end — because our 12GBs of VRAM were not enough to input 15s of audio when using a back-end. Hence, making use of domain knowledge also provides guidance for minimizing the computational cost of the model. Note that by using a single layer with many vertical and horizontal filters, one can efficiently capture the same receptive field without paying the cost of going deep. Finally, note that front-ends using many vertical and horizontal filters in the first layer are musically motivated CNNs, and these heavily rely on (musical) domain knowledge for their design.

After considering the previous discussion, we select the sample-level front-end for our assumption-free model for waveforms; and we use a spectrogram front-end with many vertical and horizontal (first-layer) filters for the model designed considering domain knowledge. Experiments below share the same back-end, which enables a fair comparison among the previously selected front-ends.

Unless otherwise stated, the following specifications are the ones used for the experiments. Throughout this chapter, we refer to these specifications as the *basic* configuration.

Shared back-end — It consists of three CNN layers (with 512 filters each and two residual connections), two pooling layers and a dense layer. In Figure 5.2 we graphically depict the shared back-end. We introduced residual connections in our model to explore very deep architectures, such that we can take advantage of the large

data available. Although adding more residual layers did not drastically improve our results, we observed that adding these residual connections stabilized learning while slightly improving performance (Li et al., 2017). The used 1D-CNN filters (Dieleman and Schrauwen, 2014) are computationally efficient and shaped such that all the features extracted by the front-end are considered across a reasonable amount of temporal context (note the 7×M' filter shapes, representing *time*×*all features*). We also make a drastic use of temporal pooling: firstly, down-sampling *x2* the temporal dimensionality of the feature maps; and secondly, by making use of global pooling with mean and max statistics. The global pooling strategy allows for variable length inputs to the network and therefore, such a model can be classified as a "variable-length input" back-end. Finally, a dense layer with 500 units connects the pooled features to a sigmoidal output.

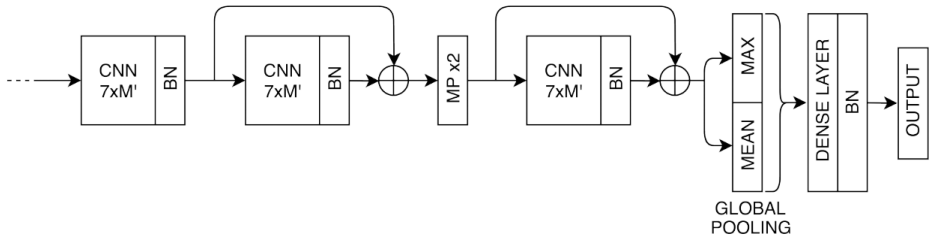


Figure 5.2: The shared back-end. *M'* stands for the feature map's vertical axis, *BN* for batch norm, and *MP* for max-pool.

Waveform front-end — It is based on a sample-level front-end (Lee et al., 2018) composed of seven stacks of: 1D-CNN (3×1 filters), batch norm, and max pool layers — see Figure 5.3. Each layer has 64, 64, 64, 128, 128, 128 and 256 filters. For the 1.2M-songs dataset, we use a model with more capacity having nine stacks with 64, 64,

64, 128, 128, 128, 128, 128, 256 filters. By hierarchically combining small-context representations and making use of max pooling, the sample-level front-end yields a feature map for an audio segment of 15s (down-sampled to 16kHz) which is further processed by the previously described back-end.

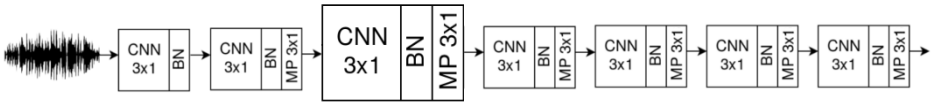


Figure 5.3: The waveform front-end. *BN* stands for batch norm, and *MP* for max-pool.

Spectrogram front-end — Firstly, audio segments are converted to log-mel magnitude spectrograms (15 seconds and 96 mel bins) and normalized to have zero-mean and unit-var. Secondly, we use vertical and horizontal filters explicitly designed to facilitate learning the timbral and temporal patterns present in spectrograms (Pons et al., 2016b; Pons and Serra, 2017; Pons et al., 2017b). Note in Figure 5.4 that the spectrogram front-end is a single-layer CNN with many filter shapes that are grouped into two branches (Pons et al., 2016b): *(i)* the top branch is designed to capture timbral features with vertical filters (Pons et al., 2017b); and *(ii)* the lower branch is designed to capture temporal features with horizontal filters (Pons and Serra, 2017). The top branch is designed to capture pitch-invariant timbral features that are occurring at different time-frequency scales in the spectrogram. Pitch invariance is enforced via enabling CNN filters to convolve through the frequency domain, and via max-pooling the feature map across its vertical axis (Pons et al., 2017b). Note that several filter shapes are used to efficiently capture

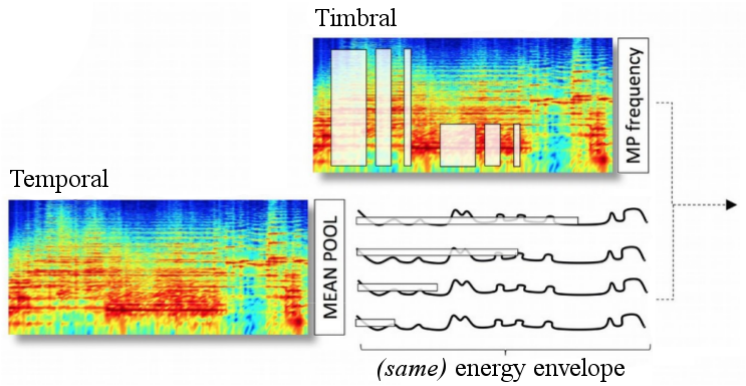


Figure 5.4: The musically motivated spectrogram front-end. *MP* stands for max-pool.

many different time-frequency patterns: 7×86, 3×86, 1×86, 7×38, 3×38 and 1×38[5] — to facilitate learning, e.g.: kick-drums (with small-rectangular filters of 7×38 capturing sub-band information for a short period of time), or string ensemble instruments (with long vertical filters of 1×86 which are capturing timbral patterns spread in the frequency axis). The lower branch is meant to learn temporal features, and is designed to efficiently capture different time-scale representations by using several long filter shapes (Pons and Serra, 2017): 165×1, 128×1, 64×1 and 32×1.[6] These filters operate over an energy envelope (not directly over the spectrogram) obtained via mean-pooling the frequency-axis of the spectrogram. By computing the energy envelope in that way, we are considering high and low frequencies together while minimizing the computations of the model. Note that no frequency/vertical convolutions are performed, but we only perform 1D (temporal) convolutions. Thus, domain knowledge is

[5]Each filter shape has 16, 32, 64, 16, 32 and 64 filters, respectively.
[6]Each filter shape has 16, 32, 64 and 128 filters, respectively.

providing guidance to minimize the computational cost of the model. The output of these two branches is concatenated, and the previously described back-end is used for going deeper.

Parameters — 50% dropout before every dense layer, ReLUs as non-linearities, and our models are trained with SGD employing Adam (with an initial learning rate of 0.001) as optimizer. We minimize the MSE for the 1.2M-songs dataset, but we minimize the cross entropy for the other datasets. During training our data are converted to audio patches of 15s, but during prediction one aims to consider the whole song. To this end, several predictions are computed for a song (by a moving window of 15s) and then averaged. Although our models are capable of predicting tags for variable-length inputs, we use fixed length patches since in preliminary experiments we observed that predicting the whole song at once yielded worse results than averaging several patch predictions. In future work we aim to further study this behavior, to find ways to exploit the fact that the whole song is generally available at inference time.

5.5 Experimental results

Along that study, we investigate how two deep learning architectures conceived considering opposite design strategies (using domain knowledge or not) perform for several datasets. First, we assess how these models scale with a dataset of 1.2M songs. Then, we study how these architectures perform for a small public dataset (MTT of $\approx$ 25k songs). And finally, we validate the proposed architectures with the biggest public dataset available (MSD of $\approx$ 240k songs).

5.5.1 1.2M-songs dataset

Experimental setup — A careful inspection of the dataset reveals that, among tags, two different data distributions dominate the annotations: *(i)* tags with bi-modal distributions, where most of the annotations are zero, which can be classified; and *(ii)* tags with pseudo-uniform distributions that can be regressed.[7] A regression tag example is *acoustic*, which indicates how acoustic a song is — from zero to one, zero being an electronic music song and one a string quartet. And a classification tag example can be any genre — for example, most songs will not be cataloged as *rap* since the dataset is large and its taxonomy contains dozens of genres.

We set as baseline a system consisting of a music feature extractor (in essence: timbre, rhythm, and harmony descriptors) and a model based on gradient boosted trees (GBT) for predicting each of the tags (Prockup et al., 2015). By predicting each tag individually, one aims to turn a hard problem into multiple (hopefully *simpler*) problems.

We use two sets of performance measurements[8]: ROC-AUC and PR-AUC for the classification tags, and error ($\sqrt{MSE}$) for the regression tags. ROC-AUC can lead to over-optimistic scores in cases where data are unbalanced (Davis and Goadrich, 2006). Given that classification tags are highly unbalanced, we also consider the PR-AUC metric since it is more indicative than ROC-AUC in these cases (Davis and Goadrich, 2006). For ROC-AUC and PR-AUC, the higher the score the better. But for $\sqrt{MSE}$, the lower the better. The studied spectrogram and waveform models are set following the *basic*

[7]Note that all output nodes are sigmoidal – i.e., we treat classification tags as regression tags for simplicity's sake.

[8]ROC: Receiver Operating Characteristic. PR: Precision Recall. AUC: Area Under the Curve. MSE: Mean Squared Error.

configuration — and are composed of 5.9M and 5.5M parameters, respectively. Given the unprecedented size of the dataset, we focus on how these models scale when trained with different amounts of data: 100k, 500k, or 1M songs. Average results (across 3 runs) are shown in Table 5.1 and in Figure 5.5.

Table 5.1: 1.2M-songs average results (3 runs) when using different training-set sizes. Baseline: GBTs+features (Prockup et al., 2015).

1.2M-songs *Models*	*train size*	*ROC AUC*	*PR AUC*	$\sqrt{MSE}$
Baseline	1.2M	91.61%	54.27%	0.1569
Waveform	1M	**92.50%**	**61.20%**	**0.1465**
Spectrogram	1M	92.17%	59.92%	0.1473
Waveform	500k	91.16%	56.42%	0.1504
Spectrogram	500k	91.61%	58.18%	0.1493
Waveform	100k	90.27%	52.76%	0.1554
Spectrogram	100k	90.14%	52.67%	0.1542

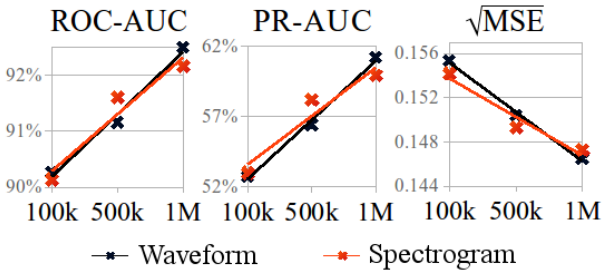


Figure 5.5: Linear regression fit on the 1.2M-songs results.

Quantitative results — Training the models with 100k songs took a few days, with 500k songs one week, and with 1M songs less than two weeks. The deep learning models trained with 1M tracks

achieve better results than the baseline in every metric. However, the deep learning models trained with 100k tracks perform worse than the baseline. This result confirms that deep learning models require large datasets to clearly outperform strong methods based on feature-design — although note that large datasets are generally not available for most audio tasks. Moreover, the biggest performance improvement *w.r.t.* the baseline is seen for PR-AUC, which provides a more informative picture of the performance when the dataset is unbalanced (Davis and Goadrich, 2006). In addition, the best performing model is based on the waveform front-end — which is capable of outperforming the spectrogram model in every metric when trained with 1M songs. This result confirms that waveform sample-level front-ends have a great potential to learn from large data, since their solution space is not constrained by any domain-knowledge inspired architectural choice. In line with that, note that the architectural choices defining the spectrogram front-end might be severely constraining the solution space. While these architectural constraints are not harmful when training data are scarce —as for the 100k/500k songs results, or in prior works (Sainath et al., 2015)—, such a strong regularization of the solution space may limit the learning capacity of the model in scenarios where large training data are available (as for the 1M songs results). One can observe this in Figure 3, where we fit linear models to the obtained results to further study this behavior. When 100k training songs are available: trend lines show that spectrogram models tend to perform better. However, when 1M training songs are available: the lines show that waveform models outperform the spectrogram ones. It is worth mentioning that the observed trends are consistent throughout metrics: ROC-AUC, PR-AUC, and $\sqrt{MSE}$.

Finally, note that there is room for improving the models under study. For example, one could address the data imbalance problem during training, or improve the back-end via exploring alternative temporal aggregation strategies.

Qualitative results — Since it is the first report of a deep music tagging model trained with such a large dataset, we also perceptually assess the quality of the estimates. To this end, we compared the predictions of one of our best performing models to the predictions of the baseline, and to the human-annotated ground-truth tags. Some interesting examples identified during this qualitative experiment are available online.[9] First, we observed that the deep learning model is biased towards predicting the popular tags (such as *lead vocals*, *English* or *male vocals*). Note that this is expected since we are not addressing the data unbalancing issue during training. And second, we observe that the baseline model (which predicts the probability of each tag with an independent GBT model) predicts mutually exclusive tags with high confidence. For example, the baseline predicted with high scores: *East Coast* & *West Coast* for an East Cost rap song, or *baroque period* & *classic period* for a Bach aria. However, the deep learning model (predicting the probability of all tags together) was able to better differentiate these similar but mutually exclusive tags. This suggests that deep learning has an advantage when compared to traditional approaches, since these mutually exclusive relations can be jointly encoded within the model.

[9] www.jordipons.me/apps/music-audio-tagging-at-scale-demo

5.5.2 MagnaTagATune dataset

Experimental setup — State-of-the-art models are set as baselines, and we use the same (classification) performance metrics as for the 1.2M-songs dataset: ROC-AUC and PR-AUC, because note that the MTT labels are binary. One of the baseline results, the SampleCNN by Lee et al. (2018) with 90.55 ROC-AUC, was computed using a slightly different version of the MTT dataset — which only includes songs having more than 1 tag and lasting more than 29.1 seconds. As a result, this cleaner version of the MTT dataset is of ≈21k songs instead of ≈25k. Although this dataset cleans out potential noisy annotations, we decided to use the original dataset to easily compare our results with former works. Thus, to fairly compare our models with the SampleCNN, we reproduce their work considering the original dataset — achieving a score of 88.56 ROC-AUC. Given that less noise is present in the SampleCNN dataset, it seems reasonable that their performance is higher than the one obtained by our implementation.

The MTT experiments can be divided in two parts: waveform and spectrogram models, see Tables 5.2 and 5.3. Due to the amenable size of the dataset (every MTT experiment lasts $<$ 5h), it is feasible to run a comprehensive study investigating different architectural configurations. Specifically, we study how waveform and spectrogram architectures behave when modifying the capacity of their front- and back-ends. For example, the experiment "# filters $\times 1/2$" in Table 5.2 consists of dividing by two the number of filters available in the waveform front-end. This means having 32, 32, 32, 64, 64, 64 and 128 filters, instead of the 64, 64, 64, 128, 128, 128 and 256 filters in the *basic* configuration. We also apply this methodology to

the spectrogram front-ends, and we add/remove capacity to them by increasing/decreasing the number of available filters. After running the front-end experiments with a fixed back-end (following the *basic* configuration: 512 CNN filters, 500 output units), we select the most promising ones to proceed with the back-end study. For waveforms, we selected: "# filters $\times 2$".[10] And for spectrograms, we selected: "# filters $\times 1/2$". Having now a fixed front-end for every experiment, we modify the capacity of the back-end via changing the number of filters in every CNN layer (512, 256, 128, 64) and changing the number of output units (500, 200). Since the *basic* configuration leads to relatively big models for the size of the dataset, these experiments explore smaller back-ends. The inputs for the MTT are set to be of 3 seconds, since longer inputs tend to yield worse results (Pons et al., 2017b; Lee et al., 2018).

Quantitative results — The waveform and spectrogram models we study outperform the proposed baselines, which are representative of the current state-of-the-art. Further, performance is quite robust to the number of parameters of the model. Although the best results are achieved by models having higher capacity, the performance difference between small and large models is minor — what means that relatively small models (which are easier to deploy) can do a reasonable job when tagging the MTT music. Finally: spectrogram models perform better than waveform models for this small public dataset. This result aligns with previous works using datasets of similar size (Pons et al., 2017b; Pons and Serra, 2017). Consequently, it confirms that domain knowledge intuitions are valid guides for designing deep architectures in scenarios where training data are scarce.

[10] "# filters $\times 2$" front-end was selected instead of "# filters $\times 4$", because it

Table 5.2: MTT results: waveform models. † Result computed with a different MTT version, see section 5.5.2.

MTT dataset *Waveform models*	*ROC AUC*	*PR AUC*	# *param*
State-of-the-art results – with our own implementations			
SampleCNN by Lee et al. (2018)†	90.55	-	2.4M
SampleCNN (reproduced)	88.56	34.38	2.4M
Dieleman and Schrauwen (2014)	84.87	-	-
Dieleman and Schrauwen (reproduced)	85.58	29.59	194k
How much capacity is required for the front-end?			
# filters ×4	**89.05**	**34.92**	11.8M
# filters ×2 (selected)	88.96	34.74	7M
# filters ×1	88.9	34.18	5.3M
# filters ×1/2	88.69	33.97	4.7M
# filters ×1/4	88.47	33.89	4.4M
How much capacity is required for the back-end? # filters in every CNN layer - # units in dense layer			
64 CNN filters - 500 units	88.57	33.99	1.3M
- 200 units	<u>88.94</u>	<u>34.47</u>	1.3M
128 CNN filters - 500 units	88.82	34.62	1.8M
- 200 units	88.81	34.6	1.7M
256 CNN filters - 500 units	88.95	34.27	3.1 M
- 200 units	88.59	34.39	2.9M
512 CNN filters - 500 units	<u>88.96</u>	<u>34.74</u>	7M
- 200 units	88.3	34.05	6.7M

Table 5.3: MTT results: spectrogram models. † Reproduced using 96 mel bands instead of 128 as in Pons et al. (2017b).

MTT dataset *Spectrogram models*	*ROC AUC*	*PR AUC*	# *param*
State-of-the-art results – with our own implementations			
VGG by Choi et al. (2016)	89.40	-	22M
VGG (reproduced)	89.99	37.56	450k
Timbre CNN by Pons et al. (2017b)	89.30	-	191k
Timbre CNN (reproduced) †	89.07	34.92	220k
How much capacity is required for the front-end?			
# filters ×1/8	90.08	37.18	4.4M
# filters ×1/4	90.12	37.69	4.6M
# filters ×1/2 (selected)	**90.40**	**38.11**	5M
# filters ×1	90.31	37.79	5.9
# filters ×2	90.07	37.29	7.6M
How much capacity is required for the back-end? # filters in every CNN layer - # units in dense layer			
64 CNN filters - 500 units	90.03	36.98	277k
- 200 units	90.28	37.55	222k
128 CNN filters - 500 units	90.16	37.61	617k
- 200 units	90.28	37.69	524k
256 CNN filters - 500 units	90.18	37.98	1.6M
- 200 units	90.06	37.16	1.4M
512 CNN filters - 500 units	90.40	38.11	5M
- 200 units	89.98	37.05	4.7M

Table 5.4: MSD results. **Top** – waveform-based models. **Bottom** – spectrogram-based models.

MSD *Models*	*ROC AUC*	*PR AUC*	# *param*
Waveform *(ours)*	87.41	28.53	5.3M
SampleCNN by Lee et al. (2018)	88.12	-	2.4M
SampleCNN multi-level & multi-scale by Lee et al. (2018)	**88.42**	-	-
Spectrogram *(ours)*	**88.75**	**31.24**	5.9M
VGG + RNN by Choi et al. (2017a)	86.2	-	3M
Multi-level & multi-scale by Lee and Nam (2017)	**88.78**	-	-

5.5.3 Million Song Dataset

Experimental setup — State-of-the-art models are set as baselines, and we use the same (classification) performance metrics as for the 1.2M-songs dataset: ROC-AUC and PR-AUC, because note that the MSD labels are binary. These experiments aim to validate the studied models with the biggest public dataset available. Models are set following the *basic* configuration, and the results are in Table 5.4.

Quantitative results — The spectrogram model outperforms the waveform model for this public dataset of ≈ 200k training songs. Furthermore, the spectrogram model performs equivalently to 'Multi-level & multi-scale' (Lee and Nam, 2017), which is the best performing method in the literature — denoting that musical knowledge can be of utility to design models for the MSD. Additionally, the waveform model performs worse than other waveform-based models that also employ sample-level front-ends. Such performance decrease

performs similarly with less parameters.

could be caused because *(i)* SampleCNN methods (Lee et al., 2018) average ten[11] estimates for the same song to compensate for possible faults in song-level predictions, while our method only averages two — via predicting consecutive patches of 15 seconds; or *(ii)* because the major difference between original SampleCNN and our waveform model is that the latter employs a global pooling strategy that could remove potentially useful information for the model. Besides, the best performing waveform-based model, the 'SampleCNN multi-level & multi-scale' by Lee et al. (2018), also achieves lower scores than the best performing spectrogram-based ones. Considering the remarkable results we report when the waveform model is trained with 1M songs, one could argue that the lack of larger public datasets is limiting the outcomes of deep learning research for music auto-tagging — particularly when dealing with end-to-end learning stacks processing raw data such as waveforms.

5.6 Summary and conclusions

This study presents the first work describing how different deep music auto-tagging architectures perform depending on the amount of available training data. We also present two architectures that yield results on par with the state-of-the-art. These architectures are based on two conceptually different design principles: one is based on a waveform front-end, and no domain knowledge inspired its design; and the other, with a spectrogram front-end, makes use of (musical) domain knowledge to justify its architectural choices. While our results suggest that models relying on domain knowledge play a rele-

[11]Since MSD audios are of 30 seconds, ten tag estimates per song can be obtained via running the model with consecutive patches of 3 seconds.

vant role in scenarios where no sizable datasets are available, we have shown that, given enough data, assumption-free models processing waveforms outperform those that rely on musical domain knowledge.

5.7 Publications, code and contributions

Our main contribution is to study, for the first time, how different deep learning architectures for music audio tagging perform when trained with a very large dataset of 1M training songs. Out of this research, a workshop and a conference article were published:

- Jordi Pons, Oriol Nieto, Matthew Prockup, Erik M. Schmidt, Andreas F. Ehmann, and Xavier Serra. *"End-to-end learning for music audio tagging at scale"*, presented at the Workshop on Machine Learning for Audio Signal Processing (ML4Audio) in NIPS 2017, and at the 19th International Society for Music Information Retrieval Conference (ISMIR2018).
  - This work was acknowledged with the best student paper award during ISMIR 2018.
  - Work done during my internship at Pandora Radio. Thanks to all my collaborators there: Oriol Nieto, Matthew Prockup, Erik M. Schmidt, and Andreas F. Ehmann. Especially, infinite thanks to Oriol Nieto — who was my mentor and close collaborator during that project.
  - The code is accessible online.[12]
  - A demonstration is also accessible online.[13]

[12] https://github.com/jordipons/music-audio-tagging-at-scale-models
[13] www.jordipons.me/apps/music-audio-tagging-at-scale-demo

Chapter 6

Audio tagging with few training data

After studying (in Chapter 5) how different deep learning architectures for music audio tagging perform when trained with a dataset of an unprecedented size, we now take a look into the low-data regime. We investigate supervised learning strategies that improve the training of neural network audio classifiers on small annotated collections. In particular, we study whether *(i)* a naive regularization of the solution space, *(ii)* prototypical networks, *(iii)* transfer learning, or *(iv)* their combination, can foster deep learning models to better leverage a small amount of training examples. To this end, we evaluate *(i–iv)* for the tasks of acoustic event recognition and acoustic scene classification, considering from 1 to 100 labeled examples per class. Results indicate that transfer learning is a powerful strategy in such scenarios, but prototypical networks show promising results when one does not count with external or validation data.

6.1 Introduction

It exists a prominent corpus of research assuming that sizable amounts of annotated audio data are available for training end-to-end classifiers (Hershey et al., 2017; Pons et al., 2018). These studies are mostly based on publicly-available datasets, where each class typically contains more than 100 audio examples (Fonseca et al., 2019, 2017; Mesaros et al., 2016; Salamon et al., 2014b). Contrastingly, only few works study the problem of training neural audio classifiers with few audio examples (Bocharov et al., 2017; Tilk, 2018; Morfi and Stowell, 2018b,a). In this work, we study how a number of neural network architectures perform in such situation. Two primary reasons motivate our work: *(i)* given that humans are able to learn novel concepts from few examples, we aim to quantify up to what extent such behavior is possible in current neural machine listening systems; and *(ii)* provided that data curation processes are tedious and expensive, it is unreasonable to assume that sizable amounts of annotated audio are always available for training neural network classifiers.

The challenge of training neural networks with few audio data has been previously addressed. For example, Morfi and Stowell (2018b) approached the problem via factorising an audio transcription task into two intermediate sub-tasks: event and tag detection. Another way to approach the problem is by leveraging additional data sources, like in unsupervised and semi-supervised frameworks where non-labelled data is also utilized (Jansen et al., 2017; Lee et al., 2009; Xu et al., 2017). Transfer learning is also a popular way to exploit such additional data sources (Kunze et al., 2017; Choi et al., 2017b), and it has been used to construct acoustic models for low-resource languages (Ghoshal et al., 2013; Huang et al., 2013), to adapt genera-

tive adversarial networks to new languages and noise types (Pascual et al., 2018), or to transfer knowledge from the visual to the audio domain (Aytar et al., 2016). An additional alternative is to use data augmentation, which has proven to be very effective for audio classification tasks (Salamon and Bello, 2017; Mun et al., 2017). However, in this work, we center our efforts into exploiting additional data resources with transfer learning. This, according to our view, has three main advantages: *(i)* differently to data augmentation, it allows leveraging external sources of data; *(ii)* it exists a rich set of techniques for learning transferable representations that one can employ (Kunze et al., 2017; Huang et al., 2013; Aytar et al., 2016); and *(iii)* transfer learning can always be further extended with data augmentation.

In parallel to previous works, the machine learning community has been developing methods for learning novel classes from few training instances, an area known as few-shot learning (Tilk, 2018; Snell et al., 2017; Ravi and Larochelle, 2016; Vinyals et al., 2016). These methods aim to build a classifier that generalizes to new classes not seen during training, given only a small number of training examples for each new class. Differently to few-shot learning, the models we study do not generalize to new classes. Instead, we assume a fixed taxonomy during both training and prediction. Still, we derive inspiration from few-shot learning for their capacity to learn from few training data. A popular approach to few-shot learning is metric learning, which aims to learn representations that preserve the class neighborhood structure so that simple distances can be measured in a learnt space (Snell et al., 2017; Vinyals et al., 2016). Such methods have been mostly used for image classification, and are very appealing due to their simplicity and yet powerful performance on several benchmarks.

In our study we consider prototypical networks (Snell et al., 2017), a metric learning approach that is based on computing distances against class-based prototypes defined in a learnt embedding space (one can think of it as a nearest-neigbour classifier trained end-to-end). Our aim is to study if prototypical networks are capable to generalize better than raw deep learning models when trained on small data. To the best of our knowledge, only Tilk (2018) has explored metric learning methods for constructing neural audio classifiers. He used the last layer features of a siamese network (Bromley et al., 1994), an alternative metric learning approach, as input to an SVM classifier. Therefore, our work can be considered the first one to employ prototypical networks for audio. Besides, back in the 80's, Kohonen (1988) proposed a distance-based model for speech recognition called learning vector quantization (LVQ), which is closely related to prototypical networks. However, LVQ is not designed to learn from few data and, furthermore, does not exploit the powerful non-linear mapping that neural networks can provide.

In this work, we investigate which strategies can provide a performance boost when neural network audio classifiers are trained with few data. These are evaluated under different low-data situations, which we describe in section 6.2. Firstly, we consider the regularization of the traditional deep learning pipeline (section 6.3.1). Next, we consider prototypical networks, with the aim to showcase the potential of metric learning-based classifiers coming from the few-shot learning literature (section 6.3.2). Finally, we consider transfer learning as a canonical way to leverage external sources of audio data (section 6.3.3). Results are presented in section 6.4 and, to conclude, we provide further discussion in section 6.5.

6.2 Methodology

6.2.1 Data: Where is the validation set?

The focus of this work is to investigate which neural network-based strategies perform best in the low-data regime. To do so, we simulate classification scenarios having only n randomly selected training audios per class, $n \in \{1, 2, 5, 10, 20, 50, 100\}$. Since results of the same repeated experiment might vary depending on which audios are selected, we run each experiment m times per fold of data, and report average accuracy scores across runs and folds. Specifically, we run the following experiments: $m = 20$ when $n \in \{1, 2\}$, $m = 10$ when $n \in \{5, 10\}$, and $m = 5$ when $n \in \{20, 50, 100\}$.

We run the study for both the tasks of acoustic event recognition and acoustic scene classification. For acoustic event recognition, we employ the UrbanSound8K dataset (US8K) by Salamon et al. (2014b), featuring 8,732 urban sounds divided into 10 classes and 10 folds (with roughly 1000 instances per class). For acoustic scene classification, we resort to the TUT dataset (ASC-TUT) by Mesaros et al. (2016, 2017), featuring 4,680 audio segments for training and 1,620 for evaluation, of 10 s each, divided into 15 classes (with 312 instances per class). Furthermore, and we consider this a crucial aspect of our work, we assume that data is so scarce that it is unreasonable to presuppose the existence of a validation set for deciding when to stop training. As a result of such constraint, the following sections also describe the rules we use to decide when to stop training. Besides reducing the train set size and not utilizing any validation set, we keep the original partitions to compare our results with previous works.

6.2.2 Baselines

To put results into context, we employ several baselines aiming to describe lower and upper bounds for the two tasks we consider:

- **Random guess:** A model picking a class at random, which scores 9.99% accuracy for US8K and 6.66% for ASC-TUT.

- **Nearest-neighbor MFCCs:** A model based on a nearest-neighbor classifier using the cosine distance over MFCC features. The feature vector is constructed from 20 MFCCs, their Δs, and $\Delta\Delta$s. We compute their mean and standard deviation through time, what results in a feature vector of size 120.

- **Salamon and Bello (2017) (SB-CNN):** This model achieves state-of-the-art results for US8K. When trained with all US8K training data it achieves an average accuracy score of 73% across folds (79% with data augmentation). SB-CNN is a model that consists of 3 convolutional layers with filters of 5×5, interleaved with max-pool layers. The resulting feature map is connected to a softmax output via a dense layer of 64 units.
  To assess how standard deep learning models would perform when learning their weights from scratch with small data, we train this baseline with different amounts of training data n.

- **Han et al. (2017b) and Mun et al. (2017):** These deep learning-based models achieve state-of-the-art performance for ASC-TUT. When trained with all ASC-TUT training data, they achieve accuracy scores of 80.4% via using an ensemble (Han et al., 2017b), and 83.3% via using an ensemble trained with GAN-based data augmentation (Mun et al., 2017).

6.2.3 Training details

Unless stated otherwise, the sections below follow this same experimental setup. Following common practice (Salamon and Bello, 2017), inputs are set to be log-mel spectrogram patches of 128 bins $\times$ 3 s (128 frames)[1]. For US8K, when audio clips are shorter than 3 s, we 'repeat-pad' spectrograms in order to meet the model's input size. That is, the original short signal is repeated up to create a 3 s signal. During training, data are randomly sampled from the original log-mel spectrograms following the previous rules. However, during prediction, if sounds are longer than 3 s, several predictions are computed by a moving window of 1 s and then averaged. We use ReLUs and a batch size of 256. Learning proceeds via minimizing the cross-entropy loss with vanilla stochastic gradient descent (SGD) at a rate of 0.1. We stabilize learning with gradient clipping, that is, we rescale the gradients so that their L2 norm does never exceed a threshold of 5.

6.3 Audio classification with few data

6.3.1 Regularized models

In a first set of experiments, we showcase the limitations of the commonly used deep learning pipeline when training data are scarce. An ordinary approach to avoid overfitting in such cases is to use regularization. Following this idea, we consider two architectures. The first one, VGG, is meant to keep the model highly expressive while introducing as much regularization as possible. The second one, TIMBRE, strongly regularizes the set of possible solutions via domain-knowledge informed architectural choices.

[1]STFT parameters: *window_size*=*hop_size*=1024 and *fs*=44.1 kHz.

- **VGG:** This is a computer vision architecture designed to make minimal assumptions regarding which are the local stationarities of the signal, so that any structure can be learnt via hierarchically combining small-context representations (Hershey et al., 2017). To this end, it is common to utilize a deep stack of small 3×3 filters (in our case 5 layers, each having only 32 filters), combined with max-pool layers (in our case of 2×2). We further employ a final dense layer with a softmax activation that adapts the feature map size to the number of output classes. We use batch norm, and ELUs as non-linearities.
- **TIMBRE:** This model is designed to learn timbral representations while keeping the model as small as possible (Pons et al., 2017b). We use a single-layer CNN with vertical filters of 108 bins $\times$ 7 frames. A softmax output is computed from the maximum values present in each CNN feature map and, therefore, the model has as many filters as output classes. TIMBRE is possibly the smallest CNN one can imagine for audio classification, provided that it only has a single 'timbral' filter per class.

Note that the studied VGG and TIMBRE models (of ≈ 50 k and 10 k parameters, respectively), are much smaller than the state-of-the-art SB-CNN model, which has ≈ 250 k parameters. Besides the regularization we introduce via minimizing the model size, we employ L2-regularization, with a penalty factor of 0.001, and make use of 50% dropout whenever a dense layer is present (only VGG and SB-CNN models have dense layers). Finally, and given that no validation set is available, we empirically find that (early) stopping training after 200 epochs provides good results for all studied models that are not based on prototypical networks.

6.3.2 Prototypical networks

In the next set of experiments, we study how prototypical networks can be exploited for audio classification with few examples. As mentioned, prototypical networks are based on learning a latent metric space in which classification can be performed by computing distances to prototype representations of each class. Prototypes μ_k are mean vectors of the embedded support data belonging to the class k. That is, given input samples x_i:

$$\mu_k = \frac{1}{|S_k|} \sum_{x_i \in S_k} f_\phi(x_i), \tag{6.1}$$

where f_ϕ is parametrized by a neural network. In our case, we use a VGG as in section 6.3.1, but with 128 filters per layer and a final linear layer instead of a softmax. The prototype vector $\mu_k \in \mathbb{R}^D$ is set to have an embedding size of $D = 10$. In this work, the support set S_k to compute each class' prototype is conformed by 5 randomly selected patches from the train set belonging to class k. These same sounds will be then reused to train f_ϕ.

Prototypical networks produce a distribution over classes for a query point x_i based on a softmax over distances to the prototypes in the embedding space:

$$p_k(x_i) = \frac{e^{-d(f_\phi(x_i), \mu_k)}}{\sum_{k'} e^{-d(f_\phi(x_i), \mu_{k'})}}, \tag{6.2}$$

where d is any suitable distance measure. We here use the Euclidean distance as we found it to outperform the cosine distance for our tasks, see Section 6.4.1. The training of f_ϕ is based on minimiz-

ing the negative log-probability of the true class via SGD. Training epochs are formed by batches of 5 random patches per class, and we backpropagate until the train set accuracy does not improve for 200 epochs. Note that this stop criteria does not utilize a validation set, only the train set. Differently from the common supervised learning pipeline, we found overfitting not to be an issue with prototypical networks — which is an important point to consider when training models with few data. Actually, monitoring how well the model is able to separate the training data in the embedding space, through measuring train set accuracy, was an effective way to assess how discriminative such space is. Although this stop criteria could promote overfitting the train set, in Section 6.4 we show that prototypical networks' generalization capabilities are still above the ones of the raw deep learning pipeline. Further discussion on the generalization capabilities of prototypical networks is available in Section 6.4.2.

6.3.3 Transfer learning

In our final set of experiments, we assess the effectiveness of canonical transfer learning strategies. For that, we use a VGG model pre-trained with Audioset (Hershey et al., 2017; Gemmeke et al., 2017), a dataset conformed by 2 M YouTube audios that was designed for training acoustic event recognition models. As a result, note that our source and target tasks are the same for US8K (all US8K classes have a direct correspondence in Audioset), but are different for ASC-TUT (only 5 out of 15 classes resemble Audioset classes). The pre-trained Audioset model[2] is composed of 6 convolutional layers with filters of 3×3, interleaved with max-pooling layers of 2×2, followed by 3 dense

[2]https://github.com/tensorflow/models/tree/master/research/audioset

layers of 4096, 4096, and 128 units. Inputs are log-mel spectrogram patches of 64 bins $\times$ 1 s (96 frames)[3]. In order to match the same conditions as previous experiments for inputs longer than 1 s, we compute several predictions by a non-overlapping moving window of 1 s that are finally averaged. When audios are shorter than 1 s, we use 'repeat-pad' as in previous experiments (see Section 6.2.3).

In order to study how the pre-trained Audioset model transfers to our tasks, we consider three alternatives:

- **Nearest-neighbor with Audioset features:** This baseline classifier serves to study how discriminative are the Audioset features alone. It is based on the cosine distance, and utilizes majority voting to aggregate the different predictions of the model through time (one per second of audio).

- **Transfer learning (fine-tuning):** The pre-trained Audioset model is fine-tuned, together with a dense softmax layer that acts as the final classifier.

- **Prototypical networks + transfer learning:** We experiment with the idea of using transfer learning in the context of prototypical networks, and we fine-tune the pre-trained model together with a dense linear layer (10 units) that defines the embedding space where the distance-based classifier operates.

For all transfer learning experiments, in order to avoid pre-trained layers to quickly overfit the train set, fine-tuning occurs at a slower pace than training the classification or embedding layer. We use a learning rate of 0.00001 for the pre-trained layers, and a learning rate of 0.1 for randomly initialized layers.

[3]STFT parameters: *window_size*=400, *hop_size*=160 and *fs*=16 kHz

6.4 Experimental results

Figure 6.1 summarizes the results we obtain for the two datasets considered in this work: US8K (first row) and ASC-TUT (second row). On the left-hand side, we compare the results of the regularized models with the ones of prototypical networks. On the right-hand side, we compare the results of transfer learning with the ones of prototypical networks. For each study case, we depict a lower bound (random guess) and an upper bound (previous state-of-the-art works using all the train set, see Section 6.2.2). In addition, we include a basic baseline consisting of a nearest neighbor classifier. All baselines and references are depicted with dashed and dotted lines.

First of all, we elaborate on the results obtained by the standard and regularized deep learning models, namely SB-CNN, VGG, and TIMBRE (Figure 6.1, left). All these models perform similarly when few training data are available ($n < 50$). Even so, it is remarkable the performance of the strongly-regularized TIMBRE model for the US8K dataset, as this outperforms the other two and the MFCC baseline when $n \leq 10$. Notice, however, that the trend changes when more data becomes avilable ($n > 20$). Under this data regime, the expressive VGG model seems to better exploit the available data. Finally, it is also interesting to observe the limitations of the commonly used deep learning pipeline when few training data are available, as the SB-CNN and regularized models struggle to clearly outperform a simple nearest-neighbor MFCC baseline for $n \leq 20$.

In order to overcome the abovementioned limitations of standard and regularized deep learning models, we investigate the use of prototypical networks. Figure 6.1 (left) depicts how they consistently out-

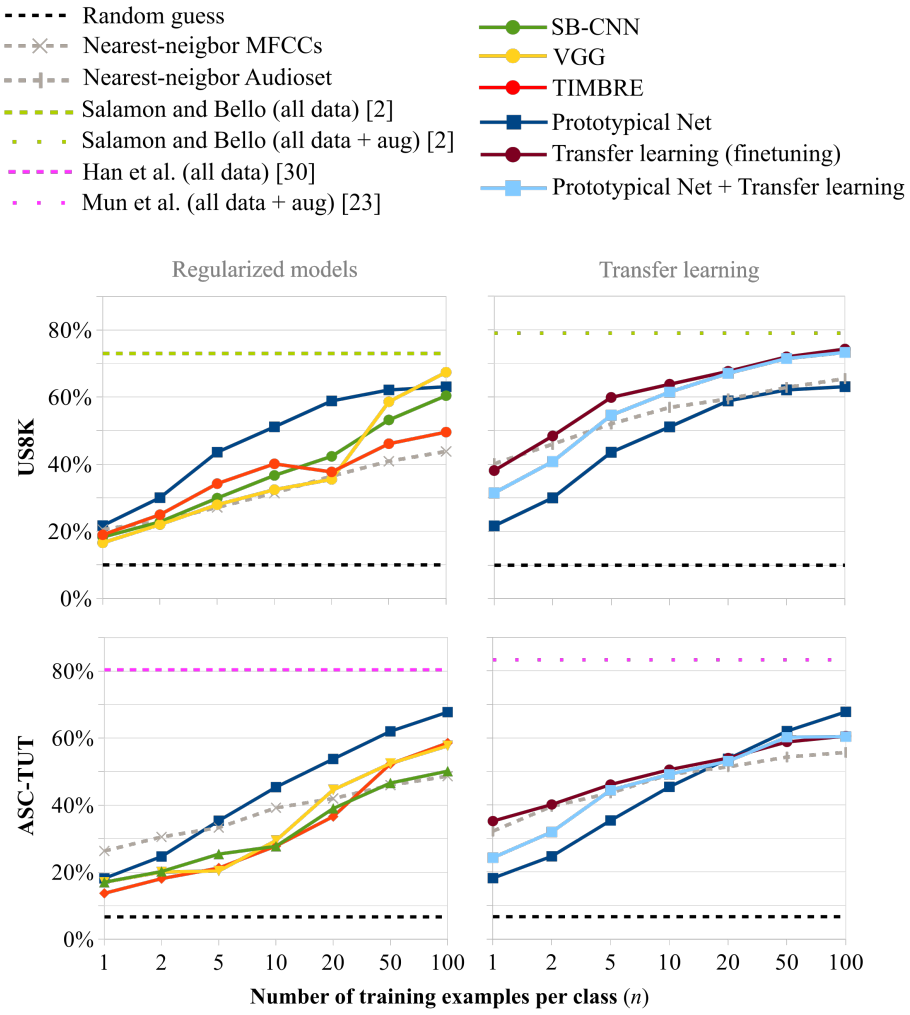


Figure 6.1: Accuracy (%) of the studied strategies when compared to prototypical networks. Dashed and dotted lines represent baselines, and strong lines represent the considered strategies. For comparison, we repeat the curve for prototypical networks in both left and right plots.

perform raw deep learning models, both for US8K and ASC-TUT. Although performance gains are moderate for $n < 5$, prototypical networks clearly outperform regularized models for $5 < n < 50$. Interestingly, though, the performance of prototypical networks can saturate for $n > 50$, see US8K results. This suggests that prototypical networks may not be as competitive as regular deep learning architectures when sizable amounts of training data are available. However, such tendency could be data-dependent, as it is not observed for ASC-TUT.

Note that the prototypical networks' embedding space is defined by a larger VGG than the regularized VGG model. Hence, prototypical networks could seem to be more prone to overfitting than regularized models. However, we find that prototypical networks do generalize better than standard and regularized models in the low data regime (Figure 6.1, left). Since overfitting seems not to dramatically affect prototypical networks' results, we find that highly expressive models (like a large VGG) can deliver good results. We speculate that this might be caused because the resulting latent space is competent enough to discriminate each of the classes, whereas for smaller and less expressive models this might not be the case.

In our study, we also investigate how transfer learning approaches compare with the previous solutions (Figure 6.1, right). We observe that, for $n \geq 10$, transfer learning with basic fine-tuning performs equivalently to prototypical networks + transfer learning. However, for $n < 10$, the former outperforms the latter. We speculate that this effect emerges when prototypes trained with small data are not representative enough of their corresponding classes. Remember that the support set to compute each class' prototype is conformed by 5 ran-

domly selected patches from the train set. As a result, for example, when $n = 1$ these 5 patches are sampled from the same spectrogram and then reused for training f_ϕ. Consequently, the variety of the training examples used for computing the prototypes can be very limited for $n < 10$, what might be harming the results of prototypical networks. Interestingly, though, for $n \geq 10$ we observe that prototypical networks + transfer learning start performing equivalently to transfer learning with fine-tuning. Note that $n = 10$ is the first scenario where prototypical networks can be trained with data being variate enough, since 5 examples can be used for computing the prototypes and 5 additional examples can be used for training f_ϕ (see section 6.3.2).

Finally, it is worth reminding that source and target tasks are the same for US8K but are different for ASC-TUT. Possibly for that reason, transfer learning consistently outperforms prototypical networks for US8K, but struggles to do so for ASC-TUT. For the latter dataset, we see that prototypical networks (trained from scratch with $n \leq 100$ instances) are able to outperform transfer learning-based approaches (pretrained with 2 M audios) for $n > 20$. This result denotes that transfer learning has a strong potential when small data are available ($n \leq 20$). However, transfer learning can be easily overthrown by prototypical networks if the number of training examples per class becomes large enough, and target and source tasks do not match.

6.4.1 Prototypical networks distance: Euclidean vs. cosine

While previous researchers (Ravi and Larochelle, 2016; Vinyals et al., 2016) employed the cosine distance for few-shot learning, the origi-

nal authors of prototypical networks found the Euclidean distance to improve their results (Snell et al., 2017). In the following, we study the impact of choosing one distance or another for the two datasets considered in our work. We report the accuracy curves for protypical networks (trained from scratch) and prototypical networks + transfer learning when trained with different amounts of data. Figure 6.2 (left) depicts US8K results, where we observe that Euclidean- and cosine-based models perform equivalently. However, for $n \geq 20$, cosine-based prototypical networks consistently achieve around 5% more accuracy. Figure 6.2 (right) depicts ASC-TUT results, where Euclidean- and cosine-based models perform similarly for $n \leq 10$. However, for $n > 10$, Euclidean-based models outperform cosine-based ones for a large margin. Consequently, for our experiments, we decide to utilize Euclidean-based prototypical networks — like the original authors of prototypical networks (Snell et al., 2017).

6.4.2 Prototypical networks: Overfitting or generalization?

One particularly interesting outcome of our work is that prototypical networks can generalize although they explicitly overfit the train set. In this section, we aim to provide further evidence of this behavior. To this end, we plot the evolution of the train set accuracy during training. For each dataset and prototypical networks-based model, we depict train set accuracy curves when training with different amounts of data. These curves (Figure 6.3) correspond to a single run and were randomly selected.

Although the test set results (depicted in Figure 6.1) clearly show that prototypical networks can generalize, it is also manifest from the

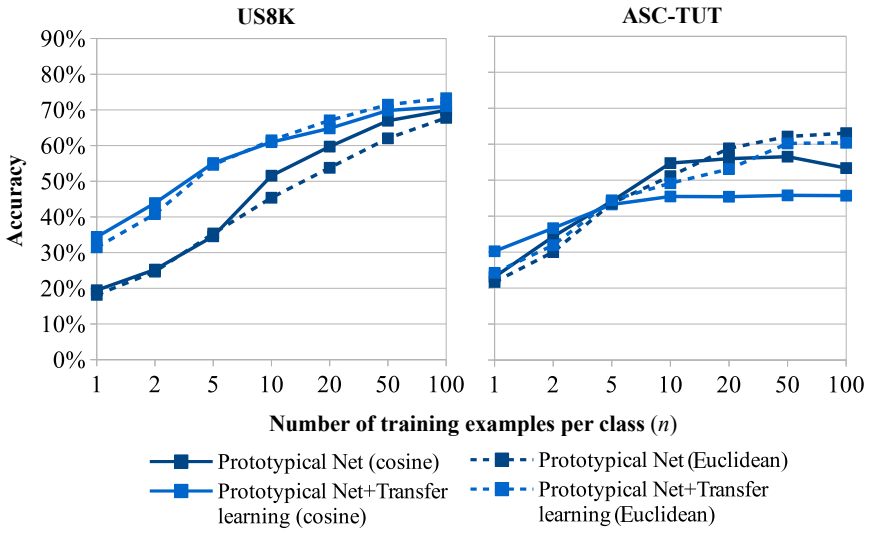


Figure 6.2: Accuracy (%) results comparing prototypical network-based models when using Euclidean or cosine distance.

train set results (depicted in Figure 6.3) that prototypical networks do overfit the train set. This effect is particularly notorious when prototypical networks are trained from scratch (no transfer learning), as these tend to quickly overfit. In addition, note that the models trained with less data ($n = 1$ or $n = 2$) tend to overfit quicker, as expected. This fact is particularly noticeable for prototypical networks + transfer learning models, since the used initialization prevents them to quickly overfit. Consequently, the training curves for $n = 1$ and $n = 2$ are more visible.

In order to further exemplify this behavior, Figure 6.4 depicts train set and test set accuracy curves when training for US8K and ASC-TUT with different amounts of data. Train set accuracy curves clearly show how prototypical networks are overfitting the train set (they achieve 100% train set accuracy). However, interestingly, test

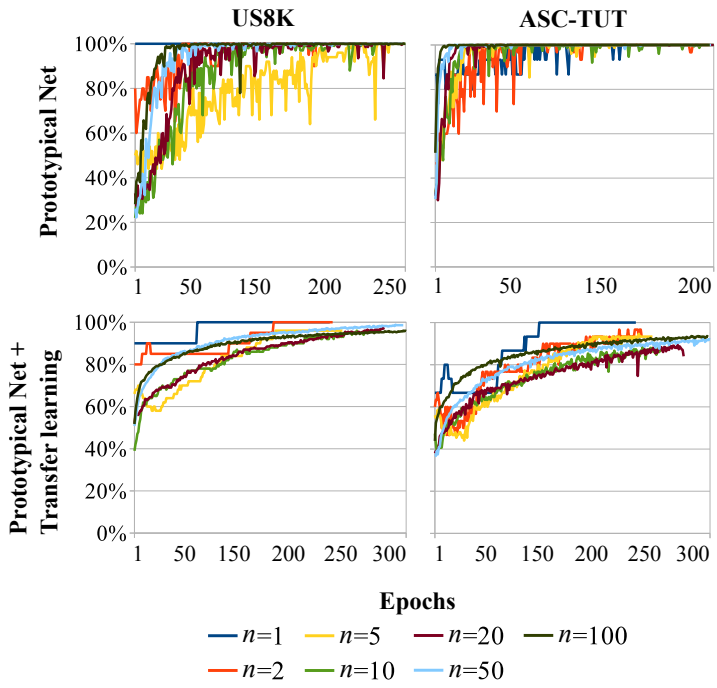


Figure 6.3: Train set accuracy results (%) for prototypical networks-based models.

set accuracy results do not decrease after the model overfits the training data. Although one would expect the performance of such models to decrease once overfitting occurs, we see that prototypical networks' performance remains unaltered.

The here described 'overfitting effect' of prototypical networks is particularly useful when only few data are available for training neural audio classifiers. As a result of such data constraints, sometimes it can be difficult to assume that a validation set is readily available. Consequently, it might be hard to know when to stop training. However, as seen, prototypical networks can overfit the train set and still

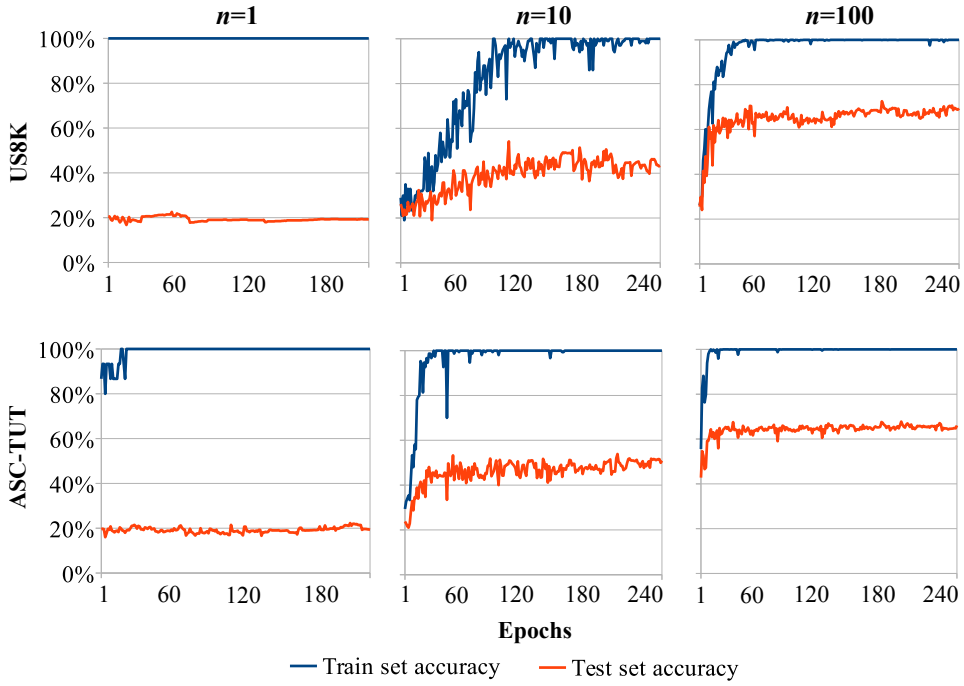


Figure 6.4: Prototypical networks' train set and test set accuracy results (%) when trained with different amounts of data (different n's).

deliver sounding results. For that reason, we propose to use the train set accuracy (measured every epoch) as a proxy to monitor how discriminative is the embedding space, to stop training after the model does not improve its train set accuracy. Note, then, that a discriminative embedding capable to separate all training examples is defined by a model explicitly overfitting the train set. Which, accordingly, would achieve 100% train set accuracy (a behavior that we repeatedly observe in Figure 6.3 and Figure 6.4). Although the stop condition we utilize is encouraging the model to explicitly overfit the train set, prototypical networks can outperform the rest of the models on the test set (see Figure 6.1). Such robustness against overfitting makes

prototypical networks particularly convenient for use cases where no validation data is accessible.

In our work, we decided to stop training after the train set accuracy did not improve for 200 epochs. It is set to 200 for consistency with the rest of the models, since these were trained for 200 epochs (see Section 6.3.1). However, note that for prototypical networks this hyper-parameter is much more robust, as for any value greater than 50 results will remain equivalent (see Figure 6.4).

6.5 Summary and conclusions

Among the strategies we have studied for training neural network classifiers with few annotated audios, we have found prototypical networks and transfer learning to be the ones providing the best results. However, choosing one or another might depend on the specificities of the use case. Transfer learning-based classifiers are generally a good choice when operating in low-data regimes, but they assume that a pre-trained model is readily available. Importantly, such model needs to be trained with data falling under a similar distribution to the few data samples we have available for solving the task. Otherwise, there is no guarantee for transfer learning to deliver better results than, for instance, prototypical networks. When data distributions do not match, we show that prototypical networks trained from scratch can be the right choice.

In order to restrict ourselves to a realistic low-data scenario, our results are computed without utilizing any validation set. As a result, the set of intuitions that generally help us deciding when to stop training no longer hold. We have found early stopping to be a valid approach when training regular deep learning classifiers. However,

interestingly, we have found overfitting not to dramatically affect prototypical networks' results — possibly because these rely on a robust distance-based classifier. Since deciding in which epoch to 'early-stop' highly depends on many design choices, we have found the 'just overfit' criteria of prototypical networks to be very simple while delivering competitive results.

6.6 Publications, code and contributions

Our main contributions are *(i)* to study how different deep learning-based strategies perform in the low data regime, and *(ii)* to propose, for the first time, the use prototypical networks to build audio classifiers. Out of this research, a conference article was published:

- Jordi Pons, Joan Serrà, and Xavier Serra (October, 2018). *"Training neural audio classifiers with few data"*, in 44th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP2019).
  - Work done in collaboration with Joan Serrà during my internship at Telefónica Research.
  - The code is accessible online.[4]

Finally, we also want to highlight a minor contribution that is closely related to the research we present along that chapter. In Appendix A, we report a negative result that we obtained when exploring the feasibility of learning the logarithmic compression of the mel spectrogram. Provided that log-mel spectrogram inputs are a widely used to build neural audio classifiers, with these experiments we were (unsuccessfully) exploring more optimal ways to pre-process the input.

[4] https://github.com/jordipons/neural-classifiers-with-few-audio

Chapter 7

Conclusions

This dissertation presents a set of experiments we designed to address the following research questions:

(i) Which deep learning architectures are most appropriate for (music) audio signals?

(ii) In which scenarios is waveform-based end-to-end learning feasible?

(iii) How much data is required for carrying out competitive deep learning research?

In the following lines, we provide additional discussion around our main findings — to conclude with a general discussion that helps understanding which are our contributions. The goal of this section is to understand the current trends and recent findings with regard to each of the topics above, to describe how our experiments contribute to the current state-of-the-art. We structure this discussion by following the above list of research questions.

7.1 Which deep learning architectures are most appropriate for (music) audio signals?

Most of our work has been centered in investigating front-ends for music and audio tagging. Hence, in the following lines, we focus on discussing how different front-ends behave when addressing audio-based classification and regression tasks. Within that context, we conclude that it currently exists an open discussion regarding the use of domain knowledge (or not) when designing deep learning models.

We structure the discussion as follows: along this section we look at how domain knowledge can be used to design spectrogram-based front-ends, to later (in Section 7.2) introduce how domain-knowledge inspired architectures have impacted the end-to-end literature.

Our main reason for exploring musically motivated CNNs (see Chapter 3) was that we found unsatisfactory to use computer vision architectures for machine listening problems. For example, we found particularly unsatisfactory to treat spectrograms as images — because images have a spatial meaning, while the spectrograms' axis stand for time and frequency. To put that in controversial terms: are we computer vision researchers working with spectrograms?

That said, note that computer vision architectures are designed considering the nature of their problem: several edges can be combined to conform a shape, and several shapes can be combined to build a nose or an eye, that can be further combined to draw a face. Many computer vision CNNs are based on this principle, and that is the reason why computer vision researchers hierarchically stack small-squared filters (Simonyan and Zisserman, 2014; He et al., 2016; Huang et al., 2017) — because these can capture edges in a first layer,

which can be hierarchically combined in deeper layers to draw a face. But modeling music and audio is about combining shapes?

Additionally, recent deep learning works on computer vision are questioning the way domain knowledge is being used to design CNNs. Motivated by the potential limitations of the CNN + max-pooling front-end, the recently proposed "capsule networks" are designed to capture the orientational and relative spatial relationships between objects in an image (Sabour et al., 2017). In that way, they can construct latent representations more robust to different view angles — what is clearly inspired by the way the human visual system works.

Natural language processing researchers have also successfully integrated domain knowledge into their designs. For example: it is usual to utilize as input a set of k-dimensional word vectors, each corresponding to the i-th word in the sentence (Kim, 2014). Considering this input and with the aim to learn n-grams, it is common to observe any of these CNN filters to span n-words. In this context, we are not aware of any hierarchical stack of small-squared CNN filters (computer-vision inspired) for processing this kind of text inputs.

Back when we started our research in 2015, we asked ourselves: why computer vision and natural language researchers have their own architectures, and the music and audio community have almost none? Is there a research opportunity that we are missing? Is this a dead end, and that is the reason why people are not publishing many music- and audio-specific architectures?

Throughout our research, we found the musically motivated CNNs to work well in practice. Not surprisingly, these models based on domain-knowledge add constraints that strongly regularize the solution space. To further understand this idea, recall that musically

motivated CNN front-ends employ vertical and horizontal filters. We included high-vertical and long-horizontal filters to constrain the solution space so that the model is encouraged to learn timbral and temporal representations from spectrograms. Although this strong regularization can be beneficial in scenarios where few training data are available, this can over-constrain the solution space in scenarios where sizable amounts of training data are available (see Chapter 5). Still, for most scenarios, musically motivated CNNs perform similarly —if not better— to its counterparts while requiring less computational resources and being more interpretable (see Chapter 3 and 4).

In addition, we found that it is not only a conceptualization issue. If not employing domain knowledge during the model design, one can run into computational costs that are possibly not necessary (see Chapters 4 and 5). To further develop this idea, in the following lines we discuss an example: to learn timbral representations with CNNs.

Considering the above reasons, it makes sense that computer vision front-ends use deep stacks of small-squared CNN filters. But to see a reasonable context considering the small-squared filters setup (e.g., the whole image), one needs to employ a rather deep model — with many layers, processing several feature maps. Provided that VRAM in GPUs is limited and each feature map representation in most computer vision CNNs takes a fairly large amount of space, it does not seem a bad idea to try to be as memory-efficient as possible.

Now let's consider the audio case, where a relevant cue is timbre (expressed along the vertical axis of a spectrogram). How to capture timbral traces using the small CNN filters front-end? Going as deep as necessary. Remember that one can only expand a small context per layer (with a small filter and a small max-pool), and to *"see"* the

whole vertical axis of a spectrogram it is required to stack several layers. However, a single CNN layer with vertical filters can already capture timbral traces (expressed along a relatively large vertical receptive field) without paying the memory cost of going deep. This happens because vertical filters already capture what is important: vertical timbral traces, and one does not need to store the output of several layers (large feature maps) to capture that context. But not only that: the number of computations required by the model utilizing vertical filters is far less than the ones performed by a computer vision model, because one just runs a single-layered front-end.[1] That said, it is important to remark that these single-layered CNN front-ends with vertical filters can work as well as front-ends based on computer vision CNNs (see Chapters 3, 4, 5, and 6). Besides, these vertical CNN filters can be designed to be pitch-invariant — what improves the model's performance (see Chapters 3, 4).

Note, then, that a very simple signal observation informed a design that makes deep learning audio models more efficient (in both time and space complexity), and this saved computing power can now be used to build more expressive models (see Chapters 3, 4 and 5).

Interestingly, it is a fact that many audio-based deep learning practitioners employ computer vision CNN front-ends for their music and audio tagging works (Choi et al., 2016, 2017b). Considering the above discussion, if more efficient and simpler audio models exist, why people keep using computer vision CNNs? The keys to our answer are: the *flexibility* of computer vision CNNs, and the *momentum* coming from the computer vision community.

[1] More discussion on learning timbral traces with CNNs in Chapter 3.4.

- The flexibility of computer vision CNNs:

Audio CNNs can be designed considering audio domain knowledge or not (for more details, see Chapter 4). And, without any doubt, spectrogram-based computer vision CNNs utilize no audio domain knowledge for their design. What is good about that? By means of not considering domain knowledge during the design, one minimizes the assumptions the model does with respect to the signal or problem. This might be beneficial, for example, if one is not certain in how to approach the task. We have observed this behavior when designing architectures for general audio tagging. While domain knowledge is a valid guide to design deep learning architectures for music tagging (see Chapter 3 and 4), we found computer vision CNNs to be very successful for general audio tagging (see Chapter 4 and 6). We hypothesize that this is caused because music has time and frequency characteristics that can be well captured with horizontal and vertical CNN filters — while environmental sounds are much more diverse in the set of features required to describe those. Provided that it can be challenging to design a specific CNN capable to encode what's relevant for general audio tagging, it seems reasonable to learn those from data with a CNN front-end that uses no audio domain knowledge for its design. With that in mind, note that part of the deep learning game is to allow the architectures to freely discover features, what leads to very successful models. If we specifically design a model to efficiently learn timbral or temporal features, we might inquire the risk of restricting too much the solution space — as it can be happening for general audio tagging (see Chapter 4 and 6).

Computer vision CNNs (based on a stack of very small filters) are designed to make minimal assumptions over the nature of the signal

or problem — so that any structure can be learnt via hierarchically combining small-context representations. Consequently, computer vision CNNs are very flexible function approximators (as opposed to be regularized models). We hypothesize that this is the reason why people use computer vision CNNs, because (in some cases) this flexibility can be useful. However, flexibility comes at a cost: these models require "larger data sets and better hardware resources" (Nam et al., 2019), and might be prone to over-fitting.

- Momentum coming from the computer vision community:

Sadly, and to put it in just a few words, the AI field tends to be over-simplified as follows: AI → deep learning → computer vision. One can find clear evidence of that in AI scientific venues, where most of the empirical results are gathered via tackling computer vision problems with deep neural networks.

Provided that the deep learning scene is clearly dominated by computer vision researchers, it seems reasonable that many exciting models, very clear tutorials, or software tools are developed towards this end. Particularly, computer vision tutorials are heavily impacting our field. For any deep learning audio practitioner, it looks easier (and possibly safer) to just follow one of these computer vision tutorials online, rather than implementing a not so well documented audio architecture. Due to that, many people end up having a computer vision model that works with *"audio images"*. Another direct consequence of this strong momentum coming from the computer vision field is that many people consider VGGs (Simonyan and Zisserman, 2014) as the *"standard CNN"*, when they are just an arbitrary design fitting the specific needs of the computer vision community.

7.2 In which scenarios is waveform-based end-to-end learning feasible?

Throughout this manuscript we show that audio domain knowledge can help improving the efficiency, interpretability, and performance of spectrogram-based models (see Chapters 3 and 4); and we also pointed out that people use spectrogram-based computer vision CNNs because these are flexible models (these are not constrained by domain knowledge). Note, then, that we have exposed the traditional discussion of using domain knowledge (or not) when designing data-driven models for the music and audio tagging use case.

Interestingly, waveform-based deep learning researchers are also diving into this discussion. Some have found very promising results when using a deep stack of very small CNN filters, but some others have found interesting results when using domain knowledge. However, the end-to-end literature is far from being conclusive. Possibly because these works are relatively new, only few independent meta-studies exist comparing these architectures across several datasets. In the following, we provide further details on which are the challenges, current trends, and recent findings in end-to-end learning research.

It is important to remark that waveforms are high-dimensional and variable. That is why, historically, the audio community did not succeed in building systems that were directly approaching raw waveforms (Dieleman and Schrauwen, 2014; Rethage et al., 2018). Just because waveforms are unintuitive and hard to approach, it possibly makes sense to tackle this problem without utilizing domain knowledge. If it is hard to know how to properly approach the task, why not learn it all from data? Accordingly, then, it could make

sense to use models relying on a deep stack of small filters. Provided that these are not constrained by any design strategy relying on domain knowledge, these are flexible models with sufficient capacity to learn from data. Moreover, when using a deep stack of small filters on top of waveforms, the possibility of learning the same representation at different phases is significantly reduced. In addition, the commonly used interleaved max-pooling layers in CNNs can further reinforce phase invariance (Lee et al., 2018; Kim et al., 2019).

As seen, it does not seem a bad idea to use waveform-based models that employ no domain knowledge for their design. Differently to the spectrogram case, since waveforms are high-dimensional and very variable, the intuitions required to design waveform-based models are not so clear as for spectrograms. As a consequence of that, researchers have proposed waveform-based models that do not rely on audio domain expertise for their design. Instead, many waveform-based models rely on a set of very small filters that can be hierarchically combined to learn any useful structure, like done for Wavenet (van den Oord et al., 2016), or for the SampleCNN (Lee et al., 2018).

Although some researchers think that utilizing no domain knowledge is the way to go for waveform-based models, some others think the contrary. All waveform-based models designed considering domain knowledge depart from the same observation: end-to-end neural networks learn frequency selective filters in the first layers (Dieleman and Schrauwen, 2014; Lee et al., 2018). If these first-layer CNNs will learn time-frequency decompositions anyway, what if one already tailors the model towards learning those? Maybe, in that way, one can achieve better results than when using a deep stack of small filters.

One first attempt towards that was to use filters that are as long

as the window length in an STFT (e.g., with a filter length of 512 samples and a stride of 256 samples). If this setup works nicely to decompose signals into sinusoidal basis with the STFT, maybe it can also work to facilitate learning frequency selective filters in a CNN (Dieleman and Schrauwen, 2014).

Later, a multiscale CNN front-end was proposed (Zhu et al., 2016). In this work, they propose to concatenate the feature maps resulting from CNNs having different filter sizes (e.g., filter lengths of 512, 256 and 128 with a stride of 64). They found that these different filters naturally learn the frequencies they can most efficiently represent, with large and small filters learning low and high frequencies respectively. This contrasts with STFT-inspired CNNs which try to cover the entire frequency spectrum with a single filter size.

Or recently, a waveform-based front-end based on parametrized sinc functions (that implement band-pass filters) was proposed. This model is called SincNet and, with just two learnable parameters in each of its first layer filters', can outperform an STFT-inspired CNN for waveforms (Ravanelli and Bengio, 2018).

Our contributions with respect to this subject are presented in Chapter 4, where we show that SampleCNN-based architectures tend to perform better than STFT-inspired ones. Although our study clearly shows that domain knowledge seems not to help when desiging waveform-based front-ends, SincNet was not yet invented when we run this study. Consequently, our study is already outdated — and we leave for future work to compare SincNet and SampleCNN front-ends.

Besides, in Chapter 5 we also investigate in which (data) scenarios waveform-based end-to-end learning is feasible. There, we contrasted a spectrogram-based model using domain knowledge (a musically mo-

tivated CNN) with a waveform-based model that is not based on domain knowledge: the SampleCNN by Lee et al. (2018). In our experiments, we have found that the expressive/flexible waveform-based SampleCNN (that is based on a deep stack of small filters) can achieve better results when training data are abundant. However, we have found that spectrogram-based models can achieve better results when less training data are accessible. We hypothesize that front-ends based on domain knowledge (including both waveform- and spectrogram-based ones) might have more chances to generalize when training data are scarce — just because the solution space is regularized, and the number of learnable parameters of a model can be reduced (like with SincNet or with musically motivated CNNs).

According to our experiments and with the current end-to-end learning models, we conclude that it is viable to develop end-to-end learning techniques for music and audio tagging — even with relatively small datasets (e.g., with the MTT dataset of $\approx$ 25k songs). However, larger datasets are required for end-to-end learning models to deliver better performance than spectrogram-based ones (see Chapter 5). Interestingly, and to confirm this trend, similar results are reported in different research areas like music source separation (Lluís et al., 2018) or lyrics to audio alignment (Stoller et al., 2019).

To conclude, spectrogram-based models tend to perform better than waveform-based ones for publicly available datasets (see Chapter 5). However, spectrogram inputs can limit the model's learning capabilities since potentially useful information (like the phase or high frequencies) tend to be discarded. In order to overcome the potential limitations associated with such pre-processing, researchers are currently exploring waveform-level front-ends, and many advances have

been made with the recent advent of deep learning (Lee et al., 2018). Given that we are in the early ages of end-to-end learning research, only few independent meta-studies exist (like the ones we presented in Chapter 4 and 5), and it is hard to tell which architectures are going to prevail in the long run. For the moment, some influent ideas were presented and now is time for the community to experiment with those.

7.3 How much data is required for carrying out competitive deep learning research?

Previously, we argued that for achieving competitive results with waveform-based models one might require large datasets. In this section, we further discuss which might the impact of having more or less training data when carrying out deep learning research.

In Table 7.1 we summarize the size of the publicly available datasets we employed for our research (sorted by size). While there are relatively small datasets, of less than 1000 audio tracks, most of the datasets are of approximately 5k–10k tracks. Besides, bigger datasets like the MagnaTagATune, the Million Song Dataset or Audioset are also available. Hence, training data does not seem to be a bottleneck for carrying out competitive auto-tagging deep learning research.

Although we conclude that data-quantity seems not to be an issue for current deep learning research, some are concerned about the quality of the data. For example, Choi et al. (2018a) studied the impact of label noise in the Million Song Dataset. Or Fonseca et al. (2017) proposed a framework for developing high-quality datasets where, e.g., annotation errors in datasets can be easily amended. Consequently, exploring the trade-off between data-quality and data-

quantity in deep learning research could also provide interesting insights for guiding the datasets annotation efforts.

Table 7.1: Size of the publicly available datasets we employed.

Dataset	# audio tracks
AudioSet (Gemmeke et al., 2017)	≈2.1M audio tracks
Million Song Dataset (Bertin-Mahieux et al., 2011)	≈240k songs
MagnaTagATune (Law et al., 2009)	≈26k songs
IRMAS (Bosch et al., 2012)	≈10k audio tracks
Urban Sound 8k (Bosch et al., 2012)	8732 audio tracks
Acoustic scene classification (Mesaros et al., 2017)	6300 audio tracks
Extended Ballroom (Marchand and Peeters, 2016)	4180 songs
GTZAN (Tzanetakis and Cook, 2002)	930 songs
Ballroom (Gouyon et al., 2004)	698 songs
Jingu a cappella singing (Black et al., 2014)	64 recordings (≈2 hours)

However, our data conclusions seem only to be true for audio tagging — where large datasets like the Million Song Dataset or Audioset are available. Recent literature on music source separation (Cano et al., 2019) and on music audio transcription (Benetos et al., 2019) have reported that the small size of the publicly available datasets is possibly limiting the outcomes of their deep learning research.

Although the data seems to be readily available, many research institutions do not have the required hardware resources (storage and computing capacity) to process it. For example, running an experiment on the Million Song Dataset requires between 1–2 weeks of (single) GPU processing. Provided that our GPU resources are limited (let's assume 2 GPUs running 24h/day), throughout a year we can only run around 50-100 experiments. To further illustrate this idea, let's now consider the Audioset dataset (of $\approx$ 2M audios). Besides the computing power required to process this data, one also needs additional hardware resources to store this large dataset in a device that can be rapidly accessible from the available GPUs.

Provided that most research institutions have few hardware resources and most datasets are of approximately 5k–10k audio tracks, most deep learning research is carried out with datasets of about that size (Han et al., 2017a; Salamon and Bello, 2017; Fonseca et al., 2018). Consequently, deep learning research with large audio datasets (>25k) and with small audio datasets (<1k) is almost unexplored.

To the best of our knowledge, only three works investigated which audio architectures perform the best with large audio datasets. One first influential work was on speech recognition (Sainath et al., 2015), a second one was on general audio tagging (Hershey et al., 2017), and we carried out some research on music audio tagging (see Chapter 5). However, most recent publications are from two years ago and it would be insightful to update these studies via also benchmarking more modern deep learning architectures — to study how these perform when trained with large audio datasets.

On the few training data side, besides the work we present in Chapter 6, we are not aware of many other works that explored how

different deep learning strategies perform when trained with few audio (Bocharov et al., 2017; Tilk, 2018; Morfi and Stowell, 2018b,a; Pons et al., 2019). While deep learning models can overfit if not trained with large datasets, investigating the possibilities of training deep neural networks with few data can enable many interesting applications. In addition, one particularly appealing aspect of this research direction is that the computational barrier is lower. Consequently: while the potential impact of this research direction is considerable, the computing power required to address this research problem is accessible for many research institutions. Besides, this research area is not yet very explored, and its potential for carrying out innovative research is remarkable. For example, a recent work by Choi et al. (2019) proposed to build audio classifiers with no training audio — formulating, in that way, the zero-shot learning problem for audio classification.

Bibliography

Adebayo, J., Gilmer, J., Goodfellow, I., and Kim, B. (2018). Local explanation methods for deep neural networks lack sensitivity to parameter values. In *International Conference on Learning Representations Workshop*.

Arandjelovic, R. and Zisserman, A. (2017). Look, listen and learn. In *IEEE International Conference on Computer Vision*, pages 609–617. IEEE.

Aytar, Y., Vondrick, C., and Torralba, A. (2016). Soundnet: Learning sound representations from unlabeled video. In *Advances in Neural Information Processing Systems*.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

Bayle, Y., Hanna, P., and Robine, M. (2017). Revisiting autotagging toward faultless instrumental playlists generation. *arXiv preprint*.

Benetos, E., Dixon, S., Duan, Z., and Ewert, S. (2019). Automatic music transcription: An overview. *IEEE Signal Processing Magazine*, 36(1):20–30.

Bengio, Y., Simard, P., Frasconi, P., et al. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bertin-Mahieux, T., Ellis, D. P., Whitman, B., and Lamere, P. (2011). The million song dataset. In *Conference of the International Society for Music Information Retrieval*.

Black, D. A., Li, M., and Tian, M. (2014). Automatic identification of emotional cues in chinese opera singing. In *International Conference on Music Perception and Cognition and Conference for the Asian-Pacific Society for Cognitive Sciences of Music*.

Bocharov, I., de Vries, A., and Tjalkens, T. (2017). K-shot learning of acoustic context. In *Neural Information Processing Systems Workshop on Machine Learning for Audio Signal Processing*.

Böck, S., Krebs, F., and Widmer, G. (2015). Accurate tempo estimation based on recurrent neural networks and resonating comb filters. In *Conference of the International Society for Music Information Retrieval*, pages 625–631.

Bosch, J. J., Janer, J., Fuhrmann, F., and Herrera, P. (2012). A comparison of sound segregation techniques for predominant instrument recognition in musical audio signals. In *Conference of the International Society for Music Information Retrieval*, pages 559–564.

Bretan, M., Oore, S., Eck, D., and Heck, L. (2017). Learning and evaluating musical features with deep autoencoders. *arXiv preprint*.

Brock, A., Donahue, J., and Simonyan, K. (2019). Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1994). Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744.

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

Cano, E., FitzGerald, D., Liutkus, A., Plumbley, M. D., and Stoter, F.-R. (2019). Musical source separation: An introduction. *IEEE Signal Processing Magazine*, 36(1):31–40.

Cano, P., Gómez, E., Gouyon, F., Herrera, P., Koppenberger, M., Ong, B., Serra, X., Streich, S., and Wack, N. (2006). ISMIR 2004 audio description contest. *Music Technology Group of the Universitat Pompeu Fabra, Technical Report*.

Chandna, P., Miron, M., Janer, J., and Gómez, E. (2017). Monoaural audio source separation using deep convolutional neural networks. In *International Conference on Latent Variable Analysis and Signal Separation*, pages 258–266. Springer.

Chen, N. and Wang, S. (2017). High-level music descriptor extraction algorithm based on combination of multi-channel cnns and lstm. In *Conference of the International Society for Music Information Retrieval*, pages 509–514.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*.

Choi, J., Lee, J., Park, J., and Nam, J. (2019). Zero-shot learning and knowledge transfer in music classification and tagging. *Machine Learning for Music Discovery Workshop at the International Conference on Machine Learning*.

Choi, K., Fazekas, G., Cho, K., and Sandler, M. (2018a). The effects of noisy labels on deep convolutional neural networks for music tagging. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(2):139–149.

Choi, K., Fazekas, G., and Sandler, M. (2016). Automatic tagging using deep convolutional neural networks. In *Conference of the International Society for Music Information Retrieval*.

Choi, K., Fazekas, G., Sandler, M., and Cho, K. (2017a). Convolutional recurrent neural networks for music classification. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 2392–2396. IEEE.

Choi, K., Fazekas, G., Sandler, M., and Cho, K. (2017b). Transfer learning for music classification and regression tasks. In *Conference of the International Society for Music Information Retrieval*.

Choi, K., Fazekas, G., Sandler, M., and Cho, K. (2018b). A comparison of audio signal preprocessing methods for deep neural networks on music tagging. In *European Signal Processing Conference*, pages 1870–1874. IEEE.

Choi, K., Fazekas, G., Sandler, M., and Kim, J. (2015). Auralisation of deep convolutional neural networks: Listening to learned features. In *Late-Breaking/Demo Session, Conference of the International Society for Music Information Retrieval*, pages 26–30.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus). *International Conference on Learning Representations*.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Dannenberg, R. B., Thom, B., and Watson, D. (1997). A machine learning approach to musical style recognition.

Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *International Conference on Machine Learning*. ACM.

Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366.

Defferrard, M., Benzi, K., Vandergheynst, P., and Bresson, X. (2017). Fma: A dataset for music analysis. In *Conference of the International Society for Music Information Retrieval*.

Dieleman, S., Brakel, P., and Schrauwen, B. (2011). Audio-based music classification with a pretrained convolutional network. In *Conference of the International Society for Music Information Retrieval*, pages 669–674.

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

Dieleman, S., van den Oord, A., and Simonyan, K. (2018). The challenge of realistic music generation: modelling raw audio at scale. In *Advances in Neural Information Processing Systems*, pages 7989–7999.

Donahue, C., McAuley, J., and Puckette, M. (2019). Adversarial audio synthesis. In *International Conference on Machine Learning*.

Durand, S., Bello, J. P., David, B., and Richard, G. (2016). Feature adapted convolutional neural networks for downbeat tracking. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 296–300. IEEE.

Eck, D. and Schmidhuber, J. (2002). Finding temporal structure in music: Blues improvisation with lstm recurrent networks. In *Workshop on Neural Networks for Signal Processing*, pages 747–756. IEEE.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Engel, J., Agrawal, K. K., Chen, S., Gulrajani, I., Donahue, C., and Roberts, A. (2019). Gansynth: Adversarial neural audio synthesis. In *International Conference on Learning Representations*.

Engel, J., Resnick, C., Roberts, A., Dieleman, S., Norouzi, M., Eck, D., and Simonyan, K. (2017). Neural audio synthesis of musical

notes with wavenet autoencoders. In *International Conference on Machine Learning*, pages 1068–1077.

Feigenbaum, E. A. (1981). Expert systems in the 1980s. *State of the art report on machine intelligence. Maidenhead: Pergamon-Infotech*.

Fonseca, E., Gong, R., and Serra, X. (2018). A simple fusion of deep and shallow learning for acoustic scene classification. In *Sound and Music Computing Conference*.

Fonseca, E., Plakal, M., Font, F., Ellis, D. P. W., Favory, X., Pons, J., and Serra, X. (2019). General-purpose tagging of freesound audio with audioset labels: task description, dataset, and baseline. In *Detection and Classification of Acoustic Scenes and Events Workshop*.

Fonseca, E., Pons, J., Favory, X., Font Corbera, F., Bogdanov, D., Ferraro, A., Oramas, S., Porter, A., and Serra, X. (2017). Freesound datasets: a platform for the creation of open audio datasets. In *Conference of the International Society for Music Information Retrieval*.

Freitag, M., Amiriparian, S., Pugachevskiy, S., Cummins, N., and Schuller, B. (2017). audeep: Unsupervised learning of representations from audio with deep recurrent neural networks. *The Journal of Machine Learning Research*, 18(1):6340–6344.

Fukushima, K. and Miyake, S. (1980). Neocognitron: Self-organizing network capable of position-invariant recognition of patterns. In *International Confonference on Pattern Recognition*, volume 1, pages 459–461.

Gemmeke, J. F., Ellis, D. P., Freedman, D., Jansen, A., Lawrence, W., Moore, R. C., Plakal, M., and Ritter, M. (2017). Audio set: An ontology and human-labeled dataset for audio events. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 776–780. IEEE.

Ghoshal, A., Swietojanski, P., and Renals, S. (2013). Multilingual training of deep neural networks. In *International Conference on Acoustics, Speech, and Signal Processing*. IEEE.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Conference on Artificial Intelligence and Statistics*, pages 249–256.

Gong, R., Pons, J., and Serra, X. (2017). Audio to score matching by combining phonetic and duration information. In *Conference of the International Society for Music Information Retrieval*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples. *arXiv preprint*.

Goto, M. and Dannenberg, R. B. (2019). Music interfaces based on automatic music signal analysis: New ways to create and listen to music. *IEEE Signal Processing Magazine*, 36(1):74–81.

Gouyon, F., Dixon, S., Pampalk, E., and Widmer, G. (2004). Evaluating rhythmic descriptors for musical genre classification. In *AES International Conference*.

Han, K., Yu, D., and Tashev, I. (2014). Speech emotion recognition using deep neural network and extreme learning machine. In *Annual Conference of the International Speech Communication Association*.

Han, Y., Kim, J., Lee, K., Han, Y., Kim, J., and Lee, K. (2017a). Deep convolutional neural networks for predominant instrument recognition in polyphonic music. *IEEE/ACM Transactions on Audio, Speech and Language Processing*, 25(1):208–221.

Han, Y., Park, J., and Lee, K. (2017b). Convolutional neural networks with binaural representations and background subtraction for acoustic scene classification. In *Detection and Classification of Acoustic Scenes and Events Workshop*, pages 1–5.

Hawthorne, C., Elsen, E., Song, J., Roberts, A., Simon, I., Raffel, C., Engel, J., Oore, S., and Eck, D. (2018). Onsets and frames: Dual-objective piano transcription. In *Conference of the International Society for Music Information Retrieval*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *International Conference on Computer Vision*, pages 1026–1034.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778.

Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. Psychology Press.

Herrera-Boyer, P., Peeters, G., and Dubnov, S. (2003). Automatic classification of musical instrument sounds. *Journal of New Music Research*, 32(1):3–21.

Hershey, S., Chaudhuri, S., Ellis, D. P., Gemmeke, J. F., Jansen, A., Moore, R. C., Plakal, M., Platt, D., Saurous, R. A., Seybold, B., et al. (2017). CNN architectures for large-scale audio classification. In *International Conference on Acoustics, Speech, and Signal Processing*. IEEE.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Holzmann, G. (2009). Reservoir computing: A powerful black-box framework for nonlinear audio processing. In *International Conference on Digital Audio Effects*.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *National Academy of Sciences*, 79(8):2554–2558.

Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Simon, I., Hawthorne, C., Shazeer, N., Dai, A. M., Hoffman, M. D., Dinculescu, M., and Eck, D. (2019). Music transformer. In *International Conference on Learning Representations*.

Huang, G., Liu, Z., Weinberger, K. Q., and van der Maaten, L. (2017). Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, page 3.

Huang, G.-B., Zhu, Q.-Y., and Siew, C.-K. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501.

Huang, J.-T., Li, J., Yu, D., Deng, L., and Gong, Y. (2013). Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *International Conference on Acoustics, Speech, and Signal Processing*. IEEE.

Huang, P.-S., Kim, M., Hasegawa-Johnson, M., and Smaragdis, P. (2015). Joint optimization of masks and deep recurrent neural networks for monaural source separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(12):2136–2147.

Humphrey, E. J. and Bello, J. P. (2012). Rethinking automatic chord recognition with convolutional neural networks. In *International Conference on Machine Learning and Applications-Volume 02*, pages 357–362. IEEE Computer Society.

Humphrey, E. J., Bello, J. P., and LeCun, Y. (2013). Feature learning and deep architectures: New directions for music informatics. *Journal of Intelligent Information Systems*, 41(3):461–481.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint*.

Jaeger, H. (2001). The echo state approach to analysing and training recurrent neural networks-with an erratum note. *German National Research Center for Information Technology, Technical Report*, 148(34):13.

Jansen, A., Plakal, M., Pandya, R., Ellis, D. P., Hershey, S., Liu, J., Moore, R. C., and Saurous, R. A. (2017). Unsupervised learning of semantic audio representations. *arXiv preprint*.

Jansson, A., Humphrey, E., Montecchio, N., Bittner, R., Kumar, A., and Weyde, T. (2017). Singing voice separation with deep u-net convolutional networks. In *Conference of the International Society for Music Information Retrieval*.

Jeong, Y., Choi, K., and Jeong, H. (2017). Dlr: Toward a deep learned rhythmic representation for music content analysis. *arXiv preprint*.

Kaminsky, I. and Materka, A. (1995). Automatic source identification of monophonic musical instrument sounds. In *International Conference on Neural Networks*, volume 1, pages 189–194. IEEE.

Kaya, H. and Salah, A. A. (2016). Combining modality-specific extreme learning machines for emotion recognition in the wild. *Journal on Multimodal User Interfaces*, 10(2):139–149.

Kereliuk, C., Sturm, B. L., and Larsen, J. (2015). Deep learning and music adversaries. *IEEE Transactions on Multimedia*, 17(11):2059–2071.

Khoo, S., Man, Z., and Cao, Z. (2012). Automatic han chinese folk song classification using extreme learning machines. In *Aus-*

tralasian Joint Conference on Artificial Intelligence, pages 49–60. Springer.

Kim, J., Urbano, J., Liem, C. C., and Hanjalic, A. (2018). One deep music representation to rule them all? a comparative analysis of different representation learning strategies. *Neural Computing and Applications*, pages 1–27.

Kim, T., Lee, J., and Nam, J. (2019). Comparison and analysis of samplecnn architectures for audio classification. *IEEE Journal of Selected Topics in Signal Processing*.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint*.

Kohonen, T. (1988). The 'neural' phonetic typewriter. *Computer*, 21(3):11–22.

Kostek, B. and Krolikowski, R. (2014). Application of artificial neural networks to the recognition of musical sounds. *Archives of Acoustics*, 22(1):27–50.

Krebs, F., Böck, S., Dorfer, M., and Widmer, G. (2016). Downbeat tracking using beat synchronous features with recurrent neural networks. In *Conference of the International Society for Music Information Retrieval*, pages 129–135.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

Kunze, J., Kirsch, L., Kurenkov, I., Krug, A., Johannsmeier, J., and Stober, S. (2017). Transfer learning for speech recognition on a budget. In *Workshop on Representation Learning for Natural Language Processing*.

Laden, B. and Keefe, D. H. (1989). The representation of pitch in a neural net model of chord classification. *Computer Music Journal*, 13(4):12–26.

Law, E., West, K., Mandel, M. I., Bay, M., and Downie, J. S. (2009). Evaluation of algorithms using games: The case of music tagging. In *Conference of the International Society for Music Information Retrieval*, pages 387–392.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

Lee, H., Pham, P., Largman, Y., and Ng, A. Y. (2009). Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in Neural Information Processing Systems*, pages 1096–1104.

Lee, J. and Nam, J. (2017). Multi-level and multi-scale feature aggregation using pretrained convolutional neural networks for music auto-tagging. *IEEE signal processing letters*, 24(8):1208–1212.

Lee, J., Park, J., Kim, K., and Nam, J. (2018). Samplecnn: End-to-end deep convolutional neural networks using very small filters for music classification. *Applied Sciences*, 8(1):150.

Lewis, J. (1988). Creation by refinement: A creativity paradigm for gradient descent learning networks. In *International Conf. on Neural Networks*, pages 229–233.

Li, H., Xu, Z., Taylor, G., and Goldstein, T. (2017). Visualizing the loss landscape of neural nets. *arXiv preprint*.

Lidy, T. (2015). Spectral convolutional neural network for music classification. *Music Information Retrieval Evaluation eX-change*.

Lluís, F., Pons, J., and Serra, X. (2018). End-to-end music source separation: is it possible in the waveform domain? In *Annual Conference of the International Speech Communication Association*.

Loh, Q.-J. B. and Emmanuel, S. (2006). Elm for the classification of music genres. In *International Conference on Control, Automation, Robotics and Vision*, pages 1–6. IEEE.

Marchand, U. and Peeters, G. (2014). The modulation scale spectrum and its application to rhythm-content analysis. In *DAFX (Digital Audio Effects)*.

Marchand, U. and Peeters, G. (2016). Scale and shift invariant time/frequency representation using auditory statistics: Application to rhythm description. In *International Workshop on Machine Learning for Signal Processing*, pages 1–6. IEEE.

Marolt, M., Kavcic, A., and Privosnik, M. (2002). Neural networks for note onset detection in piano music. In *International Computer Music Conference*.

Matityaho, B. and Furst, M. (1995). Neural network based model for classification of music type. In *Convention of Electrical and Electronics Engineers in Israel*, pages 4–3. IEEE.

McAdams, S. (2013). Musical timbre perception. *The psychology of music*, pages 35–67.

McCarthy, J., Minsky, M. L., Rochester, N., and Shannon, C. E. (1955). A proposal for the dartmouth summer research project on artificial intelligence. *AI magazine*, 27(4):12.

McFee, B., Humphrey, E. J., and Bello, J. P. (2015). A software framework for musical data augmentation. In *Conference of the International Society for Music Information Retrieval*, pages 248–254.

Mesaros, A., Heittola, T., Diment, A., Elizalde, B., Shah, A., Vincent, E., Raj, B., and Virtanen, T. (2017). Dcase 2017 challenge setup: Tasks, datasets and baseline system. In *Detection and Classification of Acoustic Scenes and Events Workshop*.

Mesaros, A., Heittola, T., and Virtanen, T. (2016). Tut database for acoustic scene classification and sound event detection. In *European Signal Processing Conference*. IEEE.

Minsky, M. and Papert, S. A. (1969). *Perceptrons: An introduction to computational geometry*. MIT press.

Miron, M., Janer Mestres, J., and Gómez Gutiérrez, E. (2017). Generating data to train convolutional neural networks for classical music source separation. In *Sound and Music Computing Conference*. Aalto University.

Morfi, V. and Stowell, D. (2018a). Data-efficient weakly supervised learning for low-resource audio event detection using deep learning. *arXiv preprint*.

Morfi, V. and Stowell, D. (2018b). Deep learning for audio transcription on low-resource datasets. *arXiv preprint*.

Mueller, M., Arzt, A., Balke, S., Dorfer, M., and Widmer, G. (2019). Cross-modal music retrieval and applications: An overview of key methodologies. *IEEE Signal Processing Magazine*, 36(1):52–62.

Mun, S., Park, S., Han, D. K., and Ko, H. (2017). Generative adversarial network based acoustic scene training set augmentation and selection using svm hyper-plane. In *Detection and Classification of Acoustic Scenes and Events Workshop*.

Nam, J., Choi, K., Lee, J., Chou, S.-Y., and Yang, Y.-H. (2019). Deep learning for audio-based music classification and tagging: Teaching computers to distinguish rock from bach. *IEEE Signal Processing Magazine*, 36(1):41–51.

Nam, J., Herrera, J., and Lee, K. (2015). A deep bag-of-features model for music auto-tagging. *arXiv preprint*.

Oramas, S., Nieto, O., Barbieri, F., and Serra, X. (2017). Multi-label music genre classification from audio, text, and images using

deep features. In *Conference of the International Society for Music Information Retrieval*.

Pao, Y.-H., Park, G.-H., and Sobajic, D. J. (1994). Learning and generalization characteristics of the random vector functional-link net. *Neurocomputing*, 6(2):163–180.

Pascual, S., Park, M., Serrà, J., Bonafonte, A., and Ahn, K.-H. (2018). Language and noise transfer in speech enhancement generative adversarial network. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 5019–5023. IEEE.

Peeters, G., Giordano, B. L., Susini, P., Misdariis, N., and McAdams, S. (2011). The timbre toolbox: Extracting audio descriptors from musical signals. *The Journal of the Acoustical Society of America*, 130(5):2902–2916.

Phan, H., Hertel, L., Maass, M., and Mertins, A. (2016). Robust audio event recognition with 1-max pooling convolutional neural networks. *arXiv preprint*.

Pons, J., Gong, R., and Serra, X. (2017a). Score-informed syllable segmentation for a cappella singing voice with convolutional neural networks. In *Conference of the International Society for Music Information Retrieval*.

Pons, J., Janer, J., Rode, T., and Nogueira, W. (2016a). Remixing music using source separation algorithms to improve the musical experience of cochlear implant users. *The Journal of the Acoustical Society of America*, 140(6):4338–4349.

Pons, J., Lidy, T., and Serra, X. (2016b). Experimenting with musically motivated convolutional neural networks. In *International Workshop on Content-Based Multimedia Indexing*, pages 1–6. IEEE.

Pons, J., Nieto, O., Prockup, M., Schmidt, E., Ehmann, A., and Serra, X. (2018). End-to-end learning for music audio tagging at scale. In *Conference of the International Society for Music Information Retrieval*.

Pons, J., Serrà, J., and Serra, X. (2019). Training neural audio classifiers with few data. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 16–20. IEEE.

Pons, J. and Serra, X. (2017). Designing efficient architectures for modeling temporal features with convolutional neural networks. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 2472–2476. IEEE.

Pons, J. and Serra, X. (2019). Randomly weighted CNNs for (music) audio classification. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 336–340. IEEE.

Pons, J., Slizovskaia, O., Gong, R., Gómez, E., and Serra, X. (2017b). Timbre analysis of music audio signals with convolutional neural networks. In *European Signal Processing Conference*, pages 2744–2748. IEEE.

Porter, A., Bogdanov, D., Kaye, R., Tsukanov, R., and Serra, X. (2015). Acousticbrainz: a community platform for gathering music information obtained from audio. In *Conference of the International Society for Music Information Retrieval*.

Prockup, M., Asman, A. J., Gouyon, F., Schmidt, E. M., Celma, O., and Kim, Y. E. (2015). Modeling rhythm using tree ensembles and the music genome project. *Machine Learning for Music Discovery Workshop at the International Conference on Machine Learning*.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Raffel, C. (2016). *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. PhD thesis, Columbia University.

Raffel, C. and Ellis, D. P. (2016). Feed-forward networks with attention can solve some long-term memory problems. In *International Conference on Learning Representations Workshop*.

Rafii, Z., Liutkus, A., Stter, F.-R., Mimilakis, S. I., and Bittner, R. (2017). The musdb18 corpus for music separation.

Ravanelli, M. and Bengio, Y. (2018). Speaker recognition from raw waveform with sincnet. In *IEEE Spoken Language Technology Workshop*, pages 1021–1028. IEEE.

Ravi, S. and Larochelle, H. (2016). Optimization as a model for few-shot learning. *International Conference on Learning Representations*.

Rethage, D., Pons, J., and Serra, X. (2018). A Wavenet for speech denoising. In *International Conference on Acoustics, Speech, and Signal Processing*. IEEE.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Roebel, A., Pons, J., Liuni, M., and Lagrangey, M. (2015). On automatic drum transcription using non-negative matrix deconvolution and itakura saito divergence. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 414–418. IEEE.

Rosenblatt, F. (1957). The perceptron: A perceiving and recognizing automation. Technical report, Cornell Aeronautical Laboratory Report.

Rosenfeld, A. and Tsotsos, J. K. (2018). Intriguing properties of randomly weighted networks: Generalizing while learning next to nothing. *arXiv preprint*.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning representations by back-propagating error. *Nature*, 323(9):533–536.

Sabour, S., Frosst, N., and Hinton, G. E. (2017). Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866.

Sainath, T. N., Weiss, R. J., Senior, A., Wilson, K. W., and Vinyals, O. (2015). Learning the speech front-end with raw waveform cldnns. In *Annual Conference of the International Speech Communication Association*.

Salamon, J. and Bello, J. P. (2017). Deep convolutional neural networks and data augmentation for environmental sound classification. *IEEE Signal Processing Letters*, 24(3):279–283.

Salamon, J., Jacoby, C., and Bello, J. P. (2014a). A dataset and taxonomy for urban sound research. In *ACM International Conference on Multimedia*, Orlando, FL, USA.

Salamon, J., Jacoby, C., and Bello, J. P. (2014b). A dataset and taxonomy for urban sound research. In *ACM-Multimedia*.

Saxe, A. M., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. Y. (2011). On random weights and unsupervised feature learning. In *International Conference on Machine Learning*, pages 1089–1096.

Scardapane, S., Comminiello, D., Scarpiniti, M., and Uncini, A. (2013). Music classification using extreme learning machines. In *International Symposium on Image and Signal Processing and Analysis*, pages 377–381. IEEE.

Scardapane, S. and Uncini, A. (2017). Semi-supervised echo state networks for audio classification. *Cognitive Computation*, 9(1):125–135.

Schlüter, J. and Böck, S. (2014). Improved musical onset detection with convolutional neural networks. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 6979–6983. IEEE.

Schlüter, J. and Grill, T. (2015). Exploring data augmentation for improved singing voice detection with neural networks. In *Confer-*

ence of the International Society for Music Information Retrieval, pages 121–126.

Schmidt, W. F., Kraaijveld, M. A., and Duin, R. P. (1992). Feed-forward neural networks with random weights. In *International Conference on Pattern Recognition*, pages 1–4. IEEE.

Schreiber, H. and Müller, M. (2018). A single-step approach to musical tempo estimation using a convolutional neural network. In *Conference of the International Society for Music Information Retrieval*.

Schreiber, H. and Müller, M. (2019). Musical tempo and key estimation using convolutional neural networks with directional filters. *arXiv preprint*.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint*.

Slaney, M., Weinberger, K., and White, W. (2008). Learning a metric for music similarity. In *Conference of the International Society for Music Information Retrieval*.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*.

Soltau, H., Schultz, T., Westphal, M., and Waibel, A. (1998). Recognition of music types. In *International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 1137–1140. IEEE.

Sordo, M., Laurier, C., and Celma, O. (2007). Annotating music collections: How content-based similarity helps to propagate labels. In *Conference of the International Society for Music Information Retrieval*.

Southall, C., Stables, R., and Hockman, J. (2017). Automatic drum transcription for polyphonic recordings using soft attention mechanisms and convolutional neural networks. In *Conference of the International Society for Music Information Retrieval*, pages 606–612.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Stevens, S. S., Volkmann, J., and Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190.

Stoller, D., Durand, S., and Ewert, S. (2019). End-to-end lyrics alignment for polyphonic music using an audio-to-character recognition model. In *International Conference on Acoustics, Speech, and Signal Processing*. IEEE.

Stoller, D., Ewert, S., and Dixon, S. (2018). Wave-u-net: A multi-scale neural network for end-to-end audio source separation. In

Conference of the International Society for Music Information Retrieval.

Stöter, F.-R., Liutkus, A., and Ito, N. (2018). The 2018 signal separation evaluation campaign. In *International Conference on Latent Variable Analysis and Signal Separation*, pages 293–305. Springer.

Sturm, B. L. (2014). A simple method to determine if a music information retrieval system is a "horse". *IEEE Transactions on Multimedia*, 16(6):1636–1644.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint*.

Tilk, B. (2018). Make SVM great again with siamese kernel for few-shot learning. *Rejected International Conference on Learning Representations submission*.

Todd, P. (1988). A sequential network design for musical applications. In *The 1988 Connectionist Models Summer School*, pages 76–84.

Tzanetakis, G. and Cook, P. (2002). Musical genre classification of audio signals. *IEEE Transactions on speech and audio processing*, 10(5):293–302.

Ulyanov, D., Vedaldi, A., and Lempitsky, V. (2018). Deep image prior. *IEEE Conference on Computer Vision and Pattern Recognition*.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu,

K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint*.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*.

Vogl, R., Dorfer, M., Widmer, G., and Knees, P. (2017). Drum transcription via joint beat and drum modeling using convolutional recurrent neural networks. In *Conference of the International Society for Music Information Retrieval*, pages 150–157.

Wessel, D. L. (1979). Timbre space as a musical control structure. *Computer music journal*, pages 45–52.

Won, M., Chun, S., and Serra, X. (2019). Visualizing and understanding self-attention based music tagging. *Machine Learning for Music Discovery Workshop at the International Conference on Machine Learning*.

Xu, Y., Huang, Q., Wang, W., Foster, P., Sigtia, S., Jackson, P. J., and Plumbley, M. D. (2017). Unsupervised feature learning based on deep models for environmental audio tagging. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(6):1230–1241.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pages 818–833. Springer.

Zhu, Z., Engel, J. H., and Hannun, A. (2016). Learning multiscale features directly from waveforms. *arXiv preprint*.

Appendix A

Negative result: learning the logarithmic compression of the mel spectrogram

Currently, successful neural audio classifiers use log-mel spectrograms as input (Hershey et al., 2017; Salamon and Bello, 2017; Dieleman and Schrauwen, 2014). Given a mel-spectrogram matrix X, the logarithmic compression is computed as follows: $f(x) = \log(\alpha \cdot X + \beta)$. Common pairs of (α, β) are $(1, \epsilon)$ or $(10000, 1)$ (Salamon and Bello, 2017; Dieleman and Schrauwen, 2014). In this section, we explore the possibility of learning (α, β). To this end, we investigate two log-mel spectrogram variants:

- **Log-learn:** The logarithmic compression of the mel spectrogram X is optimized via SGD together with the rest of the parameters of the model. We use exponential and softplus gates to control the pace of α and β, respectively. We set the initial

pre-gate values to 7 and 1, what results in out-of-gate α and β initial values of 1096.63 and 1.31, respectively.

- **Log-EPS:** We set as baseline a log-mel spectrogram which does not learn the logarithmic compression. (α, β) are set to $(1, \epsilon)$, as done by Salamon and Bello (2017) for their SB-CNN model (see Section 6.2.2 to know more about SB-CNN).

Tables A.1 and A.3 compare the results obtained by several models (see Chapter 6.3 for the details about these models) when varying the mel spectrogram compression: log-learn vs. log-EPS. We run this study for two different datasets: US8K and ASC-TUT (in Chapter 6.2 we provide details on the nature of these datasets). To clearly illustrate which are the performance gains obtained by log-learn, Tables A.2 and A.4 list the accuracy differences between log-learn and log-EPS variants. Tables A.1 and A.2 reveal that log-lean and log-EPS results are almost identical for US8K. Although it seems that log-learn can help improving the results for SB-CNN and TIMBRE architectures, for prototypical networks and VGG one can achieve worse results. For this reason, we conclude that log-learn and log-EPS results are almost equivalent for US8K. However, for ASC-TUT dataset, log-learn results are much worse than log-EPS ones. Tables A.3 and A.4 show that log-learn only improves the results of SB-CNN models when trained with little data ($1 \leq n \leq 10$), but for the rest of the models the performance decreases substantially. Accordingly, we decide not to learn the logarithmic compression of the mel spectrogram throughout the study we present in Chapter 6.

Table A.1: US8K dataset: accuracy results comparing log-EPS (standard log-mel spectrogram) & log-learn (learned log-mel spectrogram).

n	*Prototypical Net* (log-learn)	*Prototypical Net* (log-EPS)	*SB-CNN* (log-learn)	*SB-CNN* (log-EPS))
1	19.16%	**21.69**%	**19.84**%	18.29%
2	25.85%	**30.02**%	**24.77**%	22.81%
5	35.85%	**43.58**%	**32.85**%	29.89%
10	44.90%	**51.14**%	**39.75**%	36.66%
20	51.15%	**58.86**%	**45.03**%	42.34%
50	60.39%	**62.14**%	**54.94**%	53.19%
100	**65.38**%	63.08%	60.34%	**60.43**%
n	*VGG* (log-learn)	*VGG* (log-EPS)	*TIMBRE* (log-learn)	*TIMBRE* (log-EPS)
1	**18.27**%	16.58%	**21.27**%	18.97%
2	21.53%	**22.03**%	**25.95**%	24.95%
5	26.48%	**27.93**%	34.02%	**34.20**%
10	30.89%	**32.40**%	39.20%	**40.12**%
20	32.04%	**35.49**%	**42.38**%	37.70%
50	55.22%	**58.62**%	**48.40**%	46.11%
100	64.65%	**67.41**%	**50.14**%	49.57%

Table A.2: US8K dataset: log-learn accuracy gains when compared to log-EPS.

n	*Prototypical Networks*	*SB-CNN*	*VGG*	*TIMBRE*
1	**-2.53**%	1.55%	1.69%	2.30%
2	**-4.17**%	1.96%	**-0.50**%	1.00%
5	**-7.73**%	2.96%	**-1.45**%	**-0.18**%
10	**-6.24**%	3.09%	**-1.51**%	**-0.92**%
20	**-7.71**%	2.69%	**-3.45**%	4.68%
50	**-1.75**%	1.75%	**-3.40**%	2.29%
100	2.30%	**-0.09**%	**-2.76**%	0.57%

Table A.3: ASC-TUT dataset: accuracies comparing log-EPS (standard log-mel spectrogram) & log-learn (learned log-mel spectrogram).

n	*Prototypical Net* (log-learn)	*Prototypical Net* (log-EPS)	*SB-CNN* (log-learn)	*SB-CNN* (log-EPS)
1	**19.94**%	18.16%	**18.69**%	13.70%
2	**24.72**%	24.68%	**21.34**%	18.08%
5	31.77%	**35.36**%	**26.62**%	21.24%
10	39.64%	**45.39**%	**30.22**%	27.81%
20	43.28%	**53.78**%	34.87%	**36.61**%
50	45.43%	**62.03**%	45.27%	**52.32**%
100	47.70%	**67.78**%	52.59%	**58.56**%
n	*VGG* (log-learn)	*VGG* (log-EPS)	*TIMBRE* (log-learn)	*TIMBRE* (log-EPS)
1	11.07%	**17.01**%	16.06%	**17.00**%
2	11.54%	**20.05**%	19.72%	**20.21**%
5	12.47%	**20.36**%	21.88%	**25.40**%
10	13.71%	**29.45**%	24.17%	**27.74**%
20	22.51%	**44.58**%	25.09%	**39.02**%
50	37.14%	**52.46**%	31.66%	**46.61**%
100	42.53%	**57.71**%	38.35%	**50.16**%

Table A.4: ASC-TUT dataset: log-learn accuracy gains when compared to log-EPS.

n	*Prototypical Networks*	*SB-CNN*	*VGG*	*TIMBRE*
1	1.78	4.99	**-5.94**	**-0.94**
2	0.04	3.26	**-8.51**	**-0.49**
5	**-3.59**	5.38	**-7.89**	**-3.52**
10	**-5.70**	2.41	**-15.74**	**-13.93**
20	**-10.50**	**-1.74**	**-22.07**	**-13.93**
50	**-16.60**	**-7.05**	**-15.32**	**-14.95**
100	**-20.08**	**-5.97**	**-15.18**	**-11.81**