POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e della Informazione
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

MUSICAL INSTRUMENTS RECOGNITION: A TRANSFER LEARNING APPROACH

ISPG Lab

Supervisor: **Prof. Augusto Sarti**
Co-Supervisors: **Dr. Massimiliano Zanoni**
Dr. Clara Borrelli

Candidate:
Andrea Molgora
Student ID: **874735**

Academic Year 2017/2018

You should call it entropy, for two reasons. In the first place your uncertainty function has been used in statistical mechanics under that name, so it already has a name. In the second place, and more important, nobody knows what entropy really is, so in a debate you will always have the advantage.

— J. von Neumann to C. Shannon

Ringraziamenti

Anzitutto voglio ringraziare chi mi ha seguito da vicino, con pazienza e consigliandomi sapientemente le mosse giuste, per questi lunghi e difficili mesi: Max, Clara e Michele. Ringrazio tutti i compagni di corso che mi hanno accompagnato in questi anni, e quelli che hanno in generale vissuto l'univesità con me, dal primo all'ultimo. Ringrazio particolarmente Nenno, che mi è stato di enorme supporto morale in quest'ultimo periodo di lavoro. Ringrazio tutta la mia famiglia, che è sempre e comunque presente per me, per qualsiasi cosa. Ringrazio col cuore Alessandra, che incredibilmente mi sopporta da anni e che non so come mi abbia sopportato ultimamente. Ringrazio tutti i miei amici che mi hanno aiutato col sondaggio, sono insostituibili e non potrei chiedere di meglio.

Un ultimo ringraziamento va a tutte le band che mi hanno tenuto compagnia con la loro musica, in particolare: The Minneapolis Uranium Club, The Coneheads, D.L.M.I.C, Liquids, Viagra Boys, Gee Tee, Draggs, Patsy, The Sueves, The Living Eyes, The Cowboys, Tropical Trash, Dark Thougths, Hank Wood & The Hammerheads, Murderer, R. Clown, DIÄT, CIVIC, Lumpy & The Dumpers, Vintage Crop, IDLES, Rixe, Mark Cone, R.M.F.C e Powerplant.

Abstract

The advent of the digital era has made possible to access to a huge amount of musical content. Digital distributors, such as Spotify, Deezer, or Apple Music, provide a great number of musical pieces available just a click away, often organized in catalogues. However, as the amount of available music grows, it becomes more difficult for the users to search among these vast catalogues. As a consequence, there is a strong need to organize musical pieces, in order to allow the users to be able to perform effective and efficient research. As the manual annotation process would be too expensive, it is mandatory to find an automatic solution.

A meaningful description of musical pieces requires to include information about the activity of instruments playing. In this work, we present an approach for automatic musical instrument recognition. Our work employs Deep Learning Networks, mathematical models inspired by the brain functioning which try to mimic its flexibility in the learning process. These networks, which have become popular in the machine learning community, process information and are able to infer a higher level of abstraction characterization, starting from the input data.

For our work, we developed a method able to recognize 20 different musical instruments in musical pieces. It implies that, given an audio clip, we are able to extract which musical instruments are playing. In order to effectively recognize musical instruments, it is necessary to learn their characterization. To do so, Deep Learning approaches need to analyse huge amount of audio data. One of the problems encountered in the literature is the lack of enough data to learn the musical instrument characterization. To solve this problem, our approach relies on the transfer learning technique, which exploits the knowledge learned in a domain and attempts to apply it to another. We exploit a model trained on a large dataset for the task of Sound Event Detection, and apply it in the domain of musical instrument recognition. We believe the characterization learned by this model is relevant for our task, due to their similarity. In fact, the results obtained by this approach out-

perform the results of the state-of-the-art. Moreover, this approach allows us to compute the model in less time than the state-of-the-art techniques.

As the input data must be labelled for the model to learn, part of this work has been concerned in setting up a survey in order to augment the data already present in the dataset we used. After developing our technique, in order to assess its robustness, we compared it with state-of-the-art methods from the literature on different datasets.

Sommario

L'avvento dell'era digitale ha reso possibile l'accesso ad una vasta quantità di contenuti musicali. I servizi di streaming come Spotify, Deezer o Apple Music offrono una grande varietà di brani musicali a portata di click, organizzati in cataloghi. Tuttavia, con l'aumento delle dimensioni di queste librerie, diventa sempre più difficile per l'utente fare una ricerca efficiente ed efficace. Di conseguenza, c'è la necessità di organizzare i cataloghi, in modo da permettere agli utenti di cercare facilmente il contenuto di interesse. Dal momento che l'annotazione manuale risulterebbe troppo dispendiosa, è necessario automatizzare il processo.

Una descrizione significativa di un brano musicale comprende informazioni riguardanti gli strumenti musicali presenti al suo interno. In questo lavoro mostriamo il nostro approccio per l'automatizzazione del riconoscimento di strumenti musicali. Il nostro lavoro si basa sulle reti neurali denominate Deep Neural Networks, modelli matematici che sono ispirati al cervello e che ne imitano il suo funzionamento e la sua flessibilità nell'apprendimento. Queste reti, diventate molto popolari nella comunità scientifica del machine learning per via delle loro efficacia, elaborano i dati in ingresso estraendo delle caratterizzazioni ad alto livello delle informazioni contenute in essi.

Nel nostro lavoro abbiamo sviluppato un metodo capace di identificare 20 strumenti musicali differenti in un estratto musicale. Ciò significa che possiamo estrarre l'informazione su quali strumenti musicali sono attivi all'interno dell'estratto musicale. Perché questi metodi funzionino, è necessario imparare la caratterizzazione degli strumenti musicali. Per fare ciò, i metodi Deep Learning sfruttano enormi quantità di dati. Una delle problematiche trovate nella letteratura è l'assenza di una quantità adeguata di dati per imparare la caratterizzazione. Per risolvere questa problematica, nel nostro approccio utilizziamo la tecnica denominata transfer learning, che sfrutta l'apprendimento effettuato in un dominio e cerca di applicarlo in un nuovo ambito. Sfruttiamo un modello addestrato su un abbondante dataset per il compito denominato Sound Event Detection (rilevamento di eventi

sonori), e lo utilizziamo nel nostro ambito di riconoscimento di strumenti musicali. Riteniamo che la caratterizzazione imparata da questo modello sia adeguata al nostro obbiettivo, per via della loro similarità. Infatti, i risultati ottenuti con questo approccio superano quelli dello stato dell'arte. Inoltre, questo approccio ci permette di sviluppare il modello in tempistiche ridotte rispetto a quelle delle tecniche dello stato dell'arte.

Dal momento che, per addestrare il modello, i dati in ingresso devono essere annotati riguardo gli strumenti musicali attivi, parte di questo lavoro è stata dedicata all'organizzazione di un sondaggio in modo da estendere i dati già presenti nel dataset che abbiamo usato. Dopo aver sviluppato il nostro approccio, col fine di testare la sua robustezza, lo abbiamo confrontato con altri lavori della letteratura svolti su altri dataset.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As humans, we are constantly immersed in diverse and complex acoustic scenes. Everyday, walking down the street, listening to music, communicating with each other. We are expected to be able to identify sounds and navigate in these environments, using our ears as a compass [16].

Recognizing sound sources is generally a trivial task for a human. The evolution of our species and our lifestyle imprinted this skill in our kind. Recognizing sounds has been a primitive survival instinct of early mammals. Humans used their ears to distinguish a prey from a predator, or to help companions in dangerous situations. Additionally, humans trained their ability of creating and understanding sounds through the creation of complex and diverse languages, an achievement unmatched in the animal kingdom.

In the history of our species composing, playing, and listening to music are basic interactions with sounds that have always been present. In fact, music is found in different forms in every human culture, even if they developed independently. Moreover, music has phrase structure, entails learning and cultural transmission [48], which allows the art form to evolve with the succeeding generations. For the history of music, the human ability to perceive and analyse different sounds has been fundamental. Music, as is familiar to us, could not have been composed if humans did not understand the difference between solo pitches, chords, rhythm structures or timbre [13]. As the listener is more competent with music, generally his recognition abilities improve. An experienced listener might be able to distinguish classes of instruments easily, a trained musician is able to identify instruments with similar timbres, a professional composer can recognize the structure of music, being able to dissect compositions into a hierarchy of smaller components. Learning these abilities of increasing complexity allowed music to evolve alongside with us, as our civilization progressed.

Recently, the evolution of music encountered an interesting breakthrough in music distribution. The advent of musical streaming services like Apple Music[1], Deezer[2] or Spotify[3], with a vast catalogue of music libraries just a click away, changed the music fruition approach from the users. The huge amount of music that is now easily accessible might create a problem for the listener, forced to confront a trade-off between exploration and exploitation. An exploration policy might lead the user towards music he/she does not appreciate as much as the one he/she is already listening to. An exploitation policy, instead, might cause him/her to listen only to a few songs and therefore the user might be missing out on new music that he/she would potentially likes more.

However, as the analysis of large volumes of data in the digital era became infeasible for human workers, it was required to find new solutions. Hence, the Music Information Retrieval research field started. MIR is the interdisciplinary area which focus on extracting information from musical content, combining techniques of signal processing, machine learning, musical knowledge, music perception and psychology. The MIR community developed methodologies for genre recognition, source separation, music tagging, automatic playlist generation, and more.

Moreover, researchers started to define a *Music Ontology* [54] by exploiting the data accessibility. This has been a natural extension of the *Semantic Web*, a framework where web data provides semantic information, and is therefore easily linkable across different content, information applications and systems. The aim of Music Ontology is to create a large, distributed, machine-understandable environment dealing with concepts related to music production, music consumption or music recommendation. The automated extraction of data is also of great help for the Music Ontology, as it is very costly to have human annotators complete this task.

Specifically, information regarding the instruments is one of the semantic concepts that humans commonly use in musical contexts. For example, for experienced listeners and musicians, knowing the musical instruments eases a meaningful description of a musical piece, which is of extreme value for extending the aforementioned Music Ontology. However, tagging every instrument in an audio clip is not an easy task. Humans, trained for the task, need to manually listen for every second of a clip and consequently tag each recognized instrument, for a given time window. This annotation fashion is costly and prohibits to efficiently register the knowledge of the musical

[1] `https://www.apple.com/music/`
[2] `https://www.deezer.com/en/`
[3] `https://www.spotify.com`

instruments. To tackle this problem, the MIR community researched a valid approach for *automatic musical instrument recognition*.

Having automated the instrument tagging process would also open new exciting possibilities in today's music industry. For example, it could extend the percentage of labelled songs in the distributors' catalogues, granting the users to explore the libraries in a new dimension, creating new ways to search music, and enhancing the searches with input queries, metadata or structured data. This can also be done implicitly by the digital distributors, by refining the recommendation systems based on the active instruments in a song. Instrument recognition would also enhance the genre recognition task, as the musical instruments of a piece is a highly descriptive feature for the genre space [52], meaning it is easy to cluster the genres based on the active instruments. Similarly, researchers could include the musical instruments as prior knowledge in order to improve the models used for source separation and automatic music transcription, which are other fundamental MIR tasks.

Although automating this process could be perceived as easy, due to our efficiency in identifying the instruments, it is still a challenging task to automatically distinguish the timbre of the musical instruments. In fact, the physical properties of the various musical instruments, the source interference of multiple instruments playing together, the rules imposed by music composition and the perception of the resulting sounds are obstacles hard to overcome, in order to correctly execute the task. This is a different task from *source separation*, which focus on creating multiple single tracks starting from a unique mix of multiple sources. In *instrument recognition*, we focus on identifying which instruments are present and where they are active during the audio clips. The first works in the community are from 25 years ago by Kaminsky et al [40]. In the research history we can find excellent solutions in the monophonic context, both for the classification of solo notes and solo phrases of single instruments. As an example, Krishna and Sreenivas [44] experimented with a classification for solo phrases, achieving an accuracy of about 77% for instrument family and 84% for 14 individual instruments.

Unfortunately, almost of the majority of the modern music is polyphonic, with superposition of the instruments both in time and frequency, making the content extraction particularly difficult. Moreover, due to quality of the recording and style of playing, the timbre is susceptible to a considerable variance. Due to this drawbacks, the monophonic models are underperforming, and researchers moved onto the polyphonic case, where multiple approaches were investigated. Some works focus on identifying only the *predominant* instrument in a clip, which is the more easily perceived instrument

in a given time window. Others attempt to recognize each active instrument class in a single segment. In our work, we face the problem of instrument recognition in the polyphonic context, as it is the environment which better reproduce a real use-case scenario.

The majority of the literature comprises of content-based MIR methods with Machine Learning (ML) algorithms. These techniques use a set of numerical descriptors directly extracted from the audio data (the *content* of the signal) with the aid of digital signal processing techniques. Feeding those inputs to ML algorithms, the researchers are able train a self-learning model from the data. The standard ML techniques employs hand crafted features fed as input for different classifiers, then the classifiers are evaluated against the data.

This approach helps to fill the semantic gap we have between the audio signal and the perception of musical instruments. In fact, a digitalized audio signal is simply an array of numbers. In this shape, raw data do not contain any information regarding musical instruments. With the aid of ML methodologies, we are able to train a model and then test if it manages to fill the semantic gap, by mapping the data in our domain of interest. In our case, the domain of interest is the activity of musical instruments.

Recently, with the adoption of *Deep Learning* (DL) techniques [28], researchers outperformed the results of standard Machine Learning techniques. DL methodologies are a subset of Machine Learning techniques, but instead of trying to understand the mapping function between the input features and the output domain, they are based on the attempt of mimicking the behaviour of the human brain, modelling *neurons* and *connections*. These models are comprised of multiple stacked processing *layers* constituted of neurons. The first layer for example extracts a representation from the input, the second layer extracts a representation from the output of the first layer, and so on until the output layer is reached. Of course, as these models have many layers, the number of the *parameters* of the model hastily grows. To comply with these necessities, the only solution is to generate a high volume of labelled data, which we can use to *train* our network. Creating huge amount of annotated data is a problem, however. Researchers invested in creating extensive datasets to overcome this problem. Moreover, the diversity aspect of data is of extreme importance while using Deep Learning techniques, as a very biased dataset would lead the model to an over fitted representation of the data, not suitable to deal with the problem in real life conditions.

Given these assumptions, the goal of this work is to propose a model which is able to perform instrument recognition, while exploring the advan-

tages offered by *transfer learning*. As mentioned, the size of the dataset is one of the common limitations encountered by deep learning architectures, and a procedure to overcome this problem is to pre-train a model on a very large dataset, and then use the model either as an initialization or as a fixed feature extractor for the task of interest. In this case, we exploit the VGGish model [36] trained on a pre release version of YouTube-8M dataset [2] by Google, using it as a feature extractor. The YouTube-8M is not comprised only of musical instruments, but ranges from a great variety of sounds families, such as alarms, street noises, video games, conversations, and many other sound sources. The VGG feature extractor performs transformations on audio segments of 1 second, so the resulting feature vector will be of size *number of features* $\times$ *number of seconds*. The extracted features are then fed into our model, which is based on *Deep Learning* techniques.

As aforementioned, one must take great care in selecting the data for the training procedure. Different datasets are popular for the various facets of instrument recognition, such as IRMAS [6], MedleyDB [5], RWC [29] and GoodSounds [3]. An ideal dataset would be large, diverse, and freely available. The aforementioned datasets satisfy some of these criteria, but none achieves them all simultaneously. Specifically, IRMAS has a monophonic training set, MedleyDB is not freely available and have "few" songs discouraging diversity, RWC and GoodSounds are completely monophonic. In order to improve the available data for the task, Humphrey et al. recently released the OpenMIC-2018 dataset [38], which satisfies all the aforementioned constraints. The dataset consists of 20000 audio clips. Each clip is 10 seconds long, and the dataset is annotated for 20 different instruments. There are two details that are worth discussing. Fist of all, since annotating audio frame by frame for active instruments is a very costly job for a human, OpenMIC-2018 was labelled in a *weak* fashion. Weak labelling means that an instrument is annotated as present if it is active in almost a part of the 10 seconds clip, otherwise it is annotated as absent if it is *never* active during the clip. Moreover, each audio clip has been annotated only for 3 or 4 instruments out of 20, due to the difficulty of covering every instruments and every clip with surveys for humans annotators. The instruments annotated changes from clip to clip. This means that the subset of labels available for each clip changes within the dataset. This is a problem as the missing labels do not allow to formulate the problem in a complete multi-label fashion, where each clip has a known label for each class.

In order to solve the problem we set up a survey using a partition of 1000 random clips from the original OpenMIC dataset. We asked 68 participants to annotate each recognized instrument for the presented audio clips.

Proceeding in this fashion, we managed to annotate the 1000 audio clips for all the missing instrument labels. Hence, at the end, we had a completely annotated *multilabel* dataset for 20 instruments, that we used for part of the evaluation.

Therefore, our work was developed on the OpenMIC-2018 dataset, exploiting both our survey knowledge and the entire dataset with missing labels. Our results outperformed the baseline results for the dataset. Moreover, to test the robustness of the architecture, we trained it also on MedleyDB and IRMAS and we achieved better results than the state-of-the-art, proving the consistency of the transfer learning approach.

This thesis is organized as follows. In Chapter 2 we discuss the state of the art for automatic musical instrument recognition, focusing on the most popular architectures the community has seen, ranging from classic machine learning techniques to the most recent deep neural networks. After analysing the most common approaches, we discuss the theoretical foundations in Chapter 3, starting from how the sound is perceived by humans and processed by machines, and finishing with explaining the mathematical theory of our work. In Chapter 4 we describe in detail our method, its inputs, architecture and pooling procedure. In Chapter 5 we discuss our results, confronting our model within three different datasets and their experimental set-up, and explaining our procedure to achieve a completely labelled fragment of the original primary dataset through a survey. Finally, in Chapter 6 we draw conclusions about the work and we consider some possible future improvements.

Chapter 2

State of the art

In this chapter we provide an overview of the literature and we review the most effective architectures for Automatic Musical Instrument Recognition. Specifically, in section 2.1 we discuss some general rules that are relevant for the task of classification. In section 2.2 we review the techniques employed on the task of monophonic instrument recognition, which are Machine Learning approaches. In the third section 2.3 we discuss the literature on polyphonic instrument recognition. Starting from techniques already presented in the previous section, we then focus on Deep Learning techniques. Finally, in section 2.4 we briefly discuss the related task of Sound Event Detection, briefly reviewing the VGGish architecture released by Google [36], as it is fundamental for extracting our input features, and we introduce the Multiple Instance Learning framework.

2.1 General Requirements

The process of automating instrument recognition is potentially beneficial for a variety of reasons. It could greatly enhance the extraction of labelled data for the ontologies, or extend the tagged songs in the libraries of digital distributors. Due to the difficulty of the task, it is important to outline which constraints an ideal model should satisfy. In his PhD Thesis, Martin (1999) [47] postulated some basic requirements for the sound-source recognition tasks. This task is similar to ours, as in both cases we need to recognize the *sources* of sounds, which in our case are the musical instruments. Since these requirements are relevant to our problem, they provide an overview of which requirements are needed for an ideal model.
Martin lists 6 general requirements:

1. **Generalization**. The recognition of a particular instrument should

be independent from external settings such as the performer or the acoustic environment. This is crucial, as these factors will always affect the actual timbre to identify.

2. **Handle real world complexity**. As aforementioned, we cannot rely on sounds datasets that seem excessively artificially synthesized, as they will bring our classifier to fail in a real-world complex scenario.

3. **Scalability**. We may train a classifier on tens of instrument classes, but it does not address the problem of scalability. It is more important to check the *competence of approach*. Is the system capable of learning new sounds easily? Are the features extracted flexible regarding new classes of sounds? Can the features be expanded in the future? These are important questions that must be addressed, during the design phase of a model.

4. **Exhibit graceful degradation**. Another fundamental test for our classifier is to check if it is robust regarding noise and degradation. Level of ambient noise, reverberation, multiple sound sources will unavoidably distort the target sound. Degradation of performances is normal, as this happens to human perception as well. We should ensure that the classifier is able to operate in a scenario where some audio features will be likely missing.

5. **Flexible learning strategy**. Humans usually learn through both labelled and unlabelled data, as we are capable to make inference from incomplete data and draw conclusions. This comes natural to us. Ideally, a semi-supervised learning strategy that is able to exploit unlabelled data as well as labelled ones will most likely boost the performances.

6. **Real-time**. In principle, to simulate how the human perception operates, a model should be able to follow the time evolution of the audio. It might be hard to guarantee a real-time level of operation for the system, since it may be too computationally expensive to extract the features. To solve this problem, it is common to process temporal series in *batches*, and produce output at a fixed rate. In this way, outputs are updated at a known rate which simulates a real-time system, even if the outputs themselves are not continuous in time.

Moreover, Martin states that in the case of competing systems working at the same efficiency, Ockham's razor should be applied, stating that the

simplest system, which is the one that makes less assumptions, should be favoured.

2.1.1 Algorithm Design

Machine learning tasks are classified into several broad categories. In supervised learning, the algorithm builds a mathematical model from a set of data that contains both the inputs and the desired outputs. For example, if the task were identifying whether a clip contained a certain sound, the training data for a supervised learning algorithm would include clips with and without that sound (the input), and each clip would have a label (the output) indicating whether it contained or not the object. For instrument recognition, the typical system consists in a classification task trained through supervised learning. Let us define a general procedure for the algorithm design. Let us identify 4 major steps, which are the outline of the works presented in the following sections:

1. **Pre-processing**. Starting from raw data, it is common to apply transformations, such as a denoising algorithm, which removes the noisy components of a signal, or a source separation algorithm, which separates the track into a mix of single tracks, or a PCA, which reduces the dimensionality. These additional informations are added to the model as prior knowledge.

2. **Feature extraction**. From the preprocessed data, it is used to apply mathematical operations and project the input space into a more useful feature space.

3. **Classification**. This is the core step of our process. Using the features extracted from the previous step as input, the consequent step is to train a classifier to output the probability value for each instrument class.

4. **Post-processing**. Given the operation from the previous step, it is common to take the output and apply post-processing, such as thresholding the values, or applying other transformations. If still in the training phase, it is used to compare the results with the ground truth in order to assess the validity of the model.

2.2 Monophonic Instrument Recognition

First works on Instrument Recognition focused on monophonic recordings. In this context, models are facilitated by the absence of timbral source interference, which is the superposition of different sound sources at the same time instant. With this assumption, the model does not need to decide if there are multiple instruments playing, as the clip only has one source. Hence, the classification focuses on identifying a single class of instruments for the audio clip. To further ease the task, the instruments were often recorded in laboratory conditions, and researchers created datasets from these recordings. These very characteristic datasets helped researchers to study typical features for each category of instruments. During the evolution of Instrument Recognition task, researchers transitioned from pitched notes identification to solo phrases recognition, expanding the temporal resolution of the task. Here we will review how the techniques developed during the years of research.

2.2.1 Single notes Recognition

As mentioned, the community performed the initial studies on single pitched notes. Martin [47] (1999) computed features using a particular representation of the audio, which is perceptually motivated, called Mel-Frequency Cepstrum Coefficients. Then, he built a classification system on those features, using univariate Gaussian distributions. He used the model to perform a maximum likelihood classification for each instrument he investigated.

To better study the feature space, Eronen [18] (2001) employed MFCCs along with other spectral features. He fed them as input into Gaussian Mixture Model (GMM) classifiers for 31 target instruments. The GMM is a parametric classifier where the data is assumed to have a probability density function that can be modelled as a sum of *multivariate* gaussian probability density functions. Each gaussian density is called a mixture component. The results, similar to the ones of Martin, brought to the conclusion that MFCCs were the best feature for approaching the task.

In order to explore different classifiers, Essid et al. [21] (2006b) used a set of audio features found with several feature selection algorithms, and Support Vector Machines (SVM) classifiers. They worked on 10 classical instruments. A Support Vector Machine is a classifier that, given labelled training data, is able to classify the inputs by dividing them into different spaces using planes. SVMs outperformed all the baseline GMMs and longer decision length resulted in an increased accuracy, as an indicator for the

importance of musical context.

With the intent of adding contextual knowledge, Joder et al. [39] (2009) studied early and late integration of classifier decisions in combination with SVMs. For early integration, they run a statistical evaluation of features prior to classification. This approach helped to remove the outliers of the features, highlighted by low statistical validity. Late integration means they combined the classifier decisions on windows with various lengths. It allowed to better capture the temporal evolution of the music.

Exploring new feature spaces, Yu et al. [62] (2009) approached the task by using the spectrograms of audio clips as input images for the classifier, instead of MFCCs, and using k-Nearest Neighbours (k-NN) classifiers. They tested the model on seven instruments and drums. Using the k-NN algorithm, they classified the sample with a majority vote of its neighbours, with the sample being assigned to the most common class among its k nearest neighbours. In a subsequent work, Yu et al. [63] (2014) used sparse coding on frequency cepstrum with temporal pooling and scored on a 50 instruments dataset. They applied the model to polyphonic audio as well, but it underperformed as expected.

Finally, Han et al. [33] (2016) in a similar fashion used sparse coding to learn features from mel-spectrograms, and fed them to an SVM classifier. They tested the classification accuracy on 24 classes.

2.2.2 Solo phrases Recognition

In order to be aligned with the real-world cases, researchers moved onto monophonic classification of solo phrases. The motivation behind this choice is that, even if the examples were still without a mix of sources, they had the more natural *temporal evolution* one could expect from a musical clip. One of the first studies was performed by Krishna and Sreenivas [44], which employs Line Spectral Features, an alternative representation of Linear Predictive Coding features. After extracting the LSFs, they use a Gaussian Mixture Model classifier to perform instrument identification, on 14 different classes.

Subsequently, Essid et al. [19] performed classification using GMMs with MFCCs, along with Principal Component Analysis (PCA), a statistical technique used to better describe variation in a dataset, and tested on 5 instruments.

2.3 Polyphonic Instrument Recognition

During the last years, research focused on polyphonic audio, since it is closer to the real life use-case. This is a necessary problem to face, as one of the interests of the MIR community is to model classifiers able to completely automate the instrument tagging task. For example, if we are not allowed to use these systems on polyphonic sources, even for the extraction of simpler knowledge, like the presence of human voice or drums, the models do not satisfy the requirements for the construction of useful tagged musical libraries. It is also important to choose an adequate dataset. There are datasets of individual instruments, where the single tracks are mixed together to create a multi track, such as RWC [29], or real life recordings and songs, such as MedleyDB [5], OpenMIC-2018 [38], IRMAS [6] or the IOWA collection[1]. We separate the works in the literature in two categories: experiments done with conventional Machine Learning techniques such as the ones reviewed for monophonic audio, and experiments carried out with Deep Learning techniques.

2.3.1 Machine Learning Techniques

The core aspect of machine learning on polyphonic examples is its ability of a learning machine to perform accurately on new, unseen examples after having experienced a learning data set. Samples are drawn for every classes of instruments, and presented to the model as computed representations, which are called *features*. The model attempts to build a mapping from those features to the instruments domain, which is the target domain. If the model effectively learns the mapping, it is able to predict the classes of new, unseen samples. For this subsection, we review the methods that employ hand-crafted features, meaning that for each technique researchers needed to evaluate which type of feature space worked best for the model. Fuhrmann et al. [24] in his PhD. thesis divides the approaches in 3 macro categories:

- Pure pattern recognition algorithms.
- Enhanced pattern recognition algorithms.
- Template matching algorithms.

[1] `http://theremin.music.uiowa.edu/MIS.html`

Pure pattern recognition algorithms

Many of these studies attempt to directly take advantage of the knowledge extracted from the monophonic case, in order to apply it in the polyphonic case. In the majority of cases, the methodologies remain the same of the monophonic cases. They usually recognize a single *predominant* instrument or a combination of instruments. The predominant instrument is defined as one with continuous presence in a snippet of audio and is easily audible for a human listener.

Simmermacher et al. (2006) [56] performed one of the first works using these techniques. They used MFCCs as features, along with dimensionality reduction and an SVM classifier, on predominant instrument recognition. They tested their model on the IOWA collection, which contains several classical instruments recorded in studio setting.

Shortly after, Essid et al. (2006a) [20] experimented a peculiar approach where the samples were directly classified according to the overall timbre, with the derivation of a hierarchical taxonomy. By the means of an SVM classifier, they built models for the respective classes. The method had success, but only on four jazz instruments, and was not tested on larger datasets.

Little & Pardo (2008) [46] performed a first approach towards learning through weakly labelled data, where the instrument is not assumed to be continuously present. Artificial mixes were created from the 4 instruments of IOWA, in order to increase the amount of available data. Then, they used a SVM classifier, which achieved better results on phrases rather than on single polyphonic notes. The results highlight that spectro-temporal characteristics are of great importance for a correct classification of sound mixtures.

Combining the works of Little & Pardo and Joder er al. [46, 39], Fuhrmann & Herrera [25] developed a SVM classifier for 12 instruments using low-level features extracted from weakly labelled data. They separated pitched an percussive classification, and they trained a classifier on a personal collection aimed to model a real world use case.

In table 2.1 we show the summary of these techniques, where it is evident how the SVM classifiers are preferred for this approach. We show the polyphonic density (*n.s.* if the attribute is not indicated in the work), the number of instrument classes, the nature of the used samples (real or synthesized), the name of the collection (*pers.* if the collection is unreleased) and the classifier employed.

Author	Poly.	Cat.	Type	Coll.	Class.
Simmermacher et al.	4	4	real	pers.	SVM
Essid et al.	4	12	real	pers.	SVM
Little & Pardo	3	4	art. mix	IOWA	SVM
Fuhrmann et al.	10	12	real	pers.	SVM

Table 2.1: Summary of the models for **Pure pattern recognition**. *Synonyms of the header denote, among others, polyphonic density (Poly.), number of categories (Cat.), type of data used (Type), the name of the data collection (Coll.), the classification method (Class.).*

Enhanced pattern recognition algorithms

To aid the classification step, a pre-preprocessing is applied to the dataset, introducing source separation, or multi-pitch estimation as prior knowledge. Pitch and onset information are used to detect which parts of the signal are unaffected by the polyphonic interference of sound sources, so that models could work on clearer data.

Given these assumptions, Eggink & Brown (2004) [17] applied fundamental frequency estimation to a group of five instruments (Cello, Clarinet, Flute, Oboe, Violin). Then, they used a GMM classifier trained on features extracted from the spectral data of the single instruments. They created models for each instrument, and then they combined them.

Kitahara et al. (2007) [42] employed different spectral and temporal features with PCA and Linear Discriminant Analysis (LDA) for classification. Those features were used to minimise within-class variance and maximize between-class variance. This approach enhanced features to better discriminate the categories. The dataset was created from RWC [29] and they combined single instruments to create polyphonies. They tested their model on a ensemble of four instruments.

Extending the work on RWC dataset, Heittola et al. (2009) [35] employed a non-negative matrix factorization source-filter model with MFCCs and a GMM classifier. They managed to recognize frequency spectra from a mixture of signals, using prior knowledge of note events. Polyphonic mixtures of 4 seconds length with a constant number of simultaneous instruments were generated for training and testing using the samples from the RWC library.

Lastly, Barbedo & Tzanetakis (2011) [4] used majority voting on isolated individual partials to reduce the ambiguous data caused by source interference. After fundamental frequency detection, peaks are investigated

Author	Poly.	Cat.	Type	Coll.	Class.
Eggink & Brown	n.s.	5	real	pers.	GMM
Kitahara et al.	4	5	syn. MIDI	RWC	Gauss.
Heittola et al.	6	19	art. mix	RWC	GMM
Barbedo & Tzanetakis	7	25	real	pers.	Maj. vote

Table 2.2: Summary of the models for **Enhanced pattern recognition**. *Synonyms of the header denote, among others, polyphonic density (Poly.), number of categories (Cat.), type of data used (Type), the name of the data collection (Coll.), the classification method (Class.).*

to retrieve the partials, which usually are the multiples of the fundamental frequency, and then filtered. Majority votes then determines the respective partials for each fundamental, then another majority vote identifies the instrument. Results on 25 instruments are valid, but performances decay if a percussive instrument is active in the audio.

In table 2.2 we summarize the enhanced pattern recognition algorithms, where it is noticeable the impact of Gaussian Models on these methodologies.

Template matching algorithms

In this case the probability to belong to a class is extracted by evaluating the distances to abstract representations of the classes. Basically, a template for each instrument is computed and then each signal is distance-evaluated to the templates.

Cont et al. (2007) [12] used Non-Negative Matrix Factorization (NMF) decomposition to estimate the pitches of instruments. The spectrum was fed as input into the NMF algorithm. Then, templates were computed for each note of each instrument. Combining all the templates, they created a training matrix. Prediction was extracted by matching an unknown input to the training matrix.

To test this approach on RWC, Burred et al. (2010) [8] derived a template for spectro-temporal features, by applying PCA on all the spectral envelopes for a certain class. They also used a source separation algorithm to include onset and partial information in the prior. Then, it was evaluated against random mixture of notes. Finally, the distance metric was applied.

In table 2.3 we summarize these two algorithms to ease the comparison.

Author	Poly.	Cat.	Type	Coll.	Class.
Cont et al.	2	2	real mix	pers.	NMF
Burred et al.	4	5	art. mix	RWC	prob. dist.

Table 2.3: Summary of the models for **Template matching**. *Synonyms of the header denote, among others, polyphonic density (Poly.), number of categories (Cat.), type of data used (Type), the name of the data collection (Coll.), the classification method (Class.).*

Fuhrmann's model

Fuhrmann et al [24], conducting an extensive work on feature selection, proposed a model which achieved remarkable results with respect to past works. Such model exploits 92 low-level features: local energies, spectral envelope, spectral distribution and pitch-based features. After selecting the features, a Support Vector Machine classifier is used for the classification phase. The task is performed on a preliminary version of the IRMAS dataset, with over 2000 recordings lasting between 5 and 30 seconds of 11 instruments where the *predominant instrument* is annotated. This work was also important because it was the founding stone to compare several models on the same dataset, making future results more meaningful.

Bosch et al. [6] extended the work, adding source separation in the model and separating the audio into streams: *bass*, *drums*, *melody* and *other*. The separated audio is then used for the classification task, and they outperformed Fuhrmann's model.

2.3.2 The advent of Deep Learning

In the last years, Deep Learning [28] research grew exponentially in Computer Science fields such as Computer Vision. The MIR community was inevitably affected by the popularity of these methods, as we can observe from figure 2.1 and 2.2, taken from the *Awesome deep learning music* github[2]. In figure 2.1 we can observe a histogram representing the growth of *Deep Learning* articles during the last few years.

Deep Learning (DL) methodologies are based on the attempt of modelling the human brain, in order to mimic its functioning. To do so, Deep Learning use *neural networks*, which resembles the structure of the brain. In fact, these networks are made up of single nodes called *neurons*, which

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

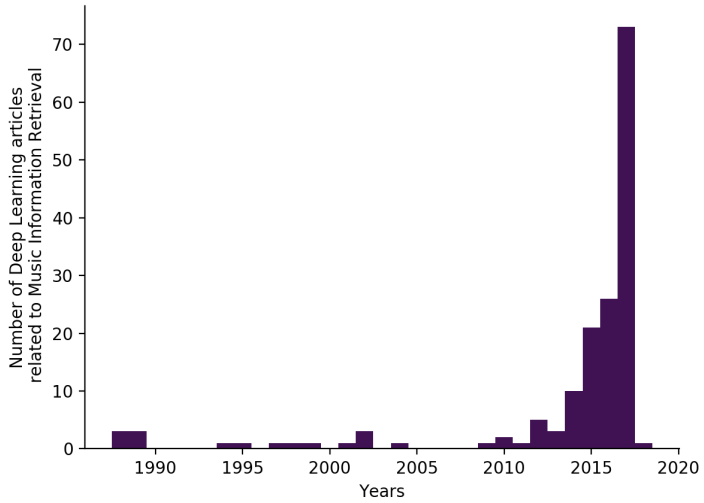


Figure 2.1: Deep Learning articles distribution in MIR, over the years. Taken from the awesome-deep-learning-music github.

have an *input* and an *output*. These nodes are organized in array-like structures called *layers*, and these layers are stacked together, one on top of the other, in order to create the *network*. These techniques were largely applied in different fields, and the introduction of *Convolutional Neural Networks* (CNNs) further increased the capability of these networks. CNNs represent a specific DL technique in which the *convolution* operation is employed. This operation, which is applied with a determined filter size, can be thought of as inspecting the input in a specific point and its surroundings, similar as a human would inspect an object by looking at it. This technique is largely employed in fields such as Computer Vision, Natural Language Processing, and Music Information Retrieval. In figure 2.2 we can observe which techniques have been employed by researchers in the last few years.

The approach of these techniques is fundamentally different from the ones of standard Machine Learning methods. ML approaches can be divided in two modules: a feature extractor and a classifier. As we have seen in the previous sections, Machine Learning techniques usually employs *hand-crafted* features, which are properly formulated for the specific task. Then, these features are fed into the classifier which learns the adequate mapping from the features into the output domain. On the other hand, Deep Learning techniques are formulated in a way that the network learns to classify input by simulating the perception of the brain. Let us allow the

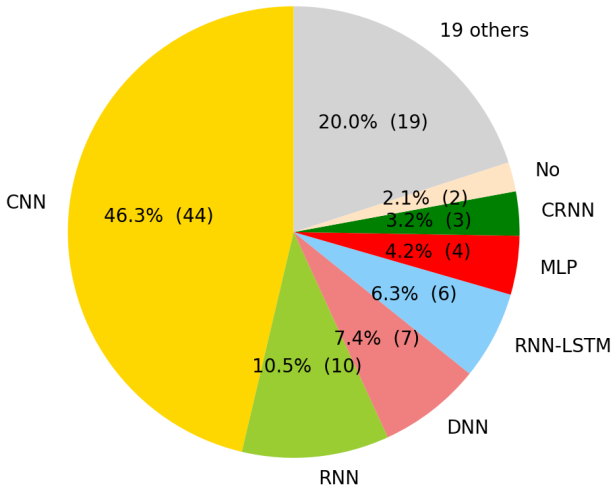


Figure 2.2: Pie chart of DL architectures in MIR. Taken from the awesome-deep-learning-music github.

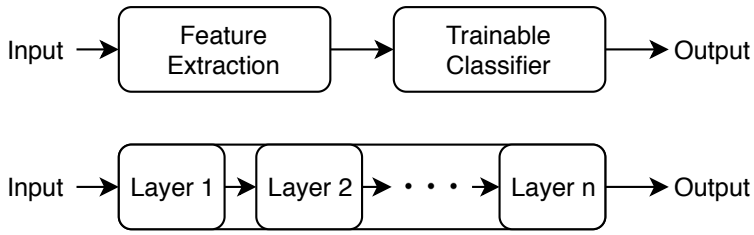


Figure 2.3: Learning procedures in ML approaches.

network to change the weights of the connections between the neurons, and repeatedly train the network on the input. This procedure is repeated until the network learns an adequate representation of the input data, by exploiting the intermediate layers which apply mathematical transformations. In figure 2.3 we summarize this comparison .

The adoption of CNNs improved results in many MIR tasks, such as onset detection [55], genre classification [14], chord estimation [66], auto tagging [15] and source separation [37].

2.3.3 Deep Learning Techniques

For the Instrument Recognition task, it is a common practice to feed the mel-spectrograms of the audio signal as inputs to Deep Learning Networks. In this way, the network can take advantage of the temporal dimension and the frequency dimension of the spectrogram. In fact, a Convolutional Neural Network can exploit the temporal proximity and the frequency distribution by inspecting those spectrograms with its convolutional filters, and applying them over the two dimensions of the input. The mel-spectrogram representation of a music signal is composed of the frequency content from the various musical instruments present in the clip. Each musical instrument produces a unique timbre, which results in a unique time-frequency spectra. A Convolutional Neural Network is therefore able to learn the instruments activity starting from the content of the mel-spectrograms. Here we discuss several works for two reasons. Firstly, we review them in detail in order to properly compare their results against ours later in chapter 5. Moreover, their overall structure is very common in the literature. In fact, the network we employ as feature extractor, which we introduce in section 2.4 and review in detail in chapter 4, is built on a very similar architecture.

Han et al. [32] proposed one of the first successful Convolutional Neural Network model for the Instrument Recognition task. They focused on the IRMAS dataset, which is already separated in a training set and a testing set. The training set consists of 6705 snippets of 3 seconds audio files, annotated for only one out of 11 instruments. The testing set is comprised of 2874 audio recordings with length varying from 5 to 20 seconds, which are labelled for more than one instrument. The instruments classes are: Cello, Clarinet, Flute, Acoustic guitar, Electric guitar, Organ, Piano, Saxophone, Trumpet, Violin and Voice.

The inputs of the network are 1 second windows mel-spectrograms. The architecture is summarized in figure 2.4. The model follows the architecture of popular computer vision models such as AlexNet [45] and VGGNet [64]. Small 3×3 convolutional filters are used, with a LeakyRelU activation function followed by a 3×3 max pooling layer. Then, a Dropout layer of 25% is applied. This block repeats 4 times, with filter depths increasing from 32 to 256. With the last set of convolutions the output is flattened and a Dense layer with 1024 units is used, followed by a Dropout layer of 50%. Sigmoidal activation is used to transform the values into the probability range.

For the evaluation phase, since the multi-label setting of the dataset is present only in the testing set, for each audio clip they summed all the outputs class-wise over the entire audio clip and they normalized by the

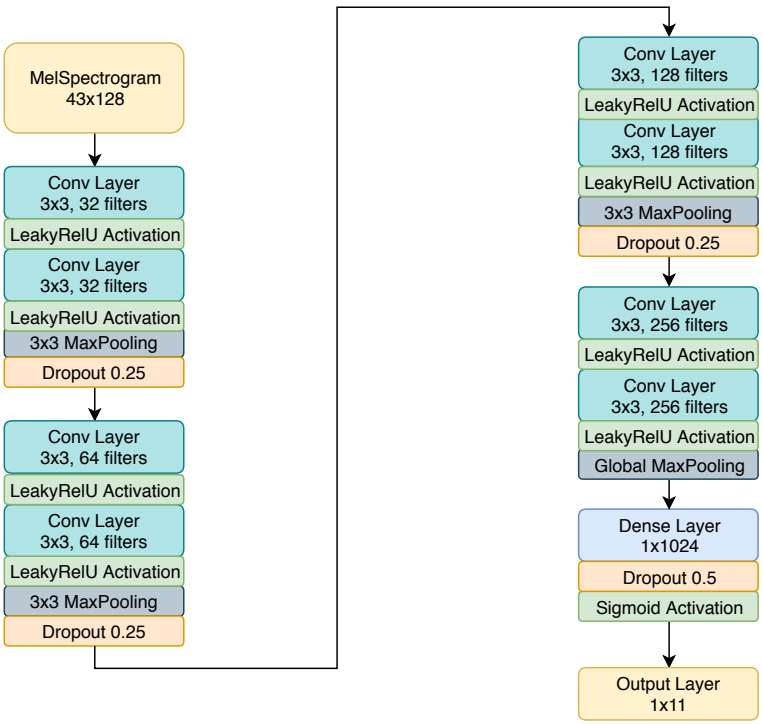


Figure 2.4: Han et al. [32] model architecture

maximum present value, in order to have values between 0 and 1. Then, they found the optimal threshold to binarize the outputs of the classifier. This method is based on the assumption that humans perceive the *predominant* instrument such that the strongest instrument is always detected, and the existence of other instruments is judged by their relative strength compared to the predominant instrument. We will also evaluate our model in this fashion, for classification carried out on this same dataset.

Following this work, another interesting experiment was performed by Pons et al. [53] on the IRMAS dataset. They investigated if it is possible to achieve better results with more efficient CNNs, using less parameters. They proposed to use different shape for the *kernel* size of the Convolutional Layers. In fact, the usage of *square* filters was predominantly adopted from the tasks of Computer Vision, where local proximity in both direction has the same meaning, domain wise. However, with mel-spectrograms, the problem extended in 2 different dimensions, frequency and time. With these assumptions, they proposed to use small rectangular $m \times n$ filters, with m being the frequency bins and n being the temporal window. For example, using a filter very narrow in time but wide in frequency provided a tempo-invariant

	Micro			
Author	Precision	Recall	F1	n. of Parameters
Bosch et al.	0.504	0.501	0.503	-
Han et al.	**0.655**	**0.557**	**0.602**	1446k
Pons - Single	0.611	0.516	0.559	62k
Pons - Multi	0.650	0.538	0.589	743k

Table 2.4: Summary of the models on the IRMAS Dataset.

view of the spectra, while a very narrow filter in frequency but large in time provided the partials evolution of a certain frequency bin over time.

However, since there is no solution to which one of the filters they should choose, they decided to employ a tower-like architecture where for the first layer multiple filters were used, and the results are concatenated. They proposed 2 different architectures:

- **Single-layer** has a single but wide convolutional layer with filters of various sizes. They used 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. They max-pooled the m dimension to learn pitch invariant representations, using max pooling in an area (m,16). 50% dropout is applied to the 11- way softmax output layer. Each convolutional layer is followed by Batch Normalization.

- **Multi-layer** architecture: first layer has the same settings as single-layer but it is deepen 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 and Batch Normalization.

For the input, they computed mel-spectrograms with 96 frequency bins. Evaluation was performed as the same way by Han et al.

With a considerably smaller architecture, they achieved remarkable results, summarized in the table 2.4.
Another approach was made by Gururani et al. (2018) [31]. They investigated a new dataset made by combining MedleyDB [5] and Mixing-Secrets [30]. The resulting set is optimal for the multi-label identification problem, as it has multi-label annotations for both the training set and the testing

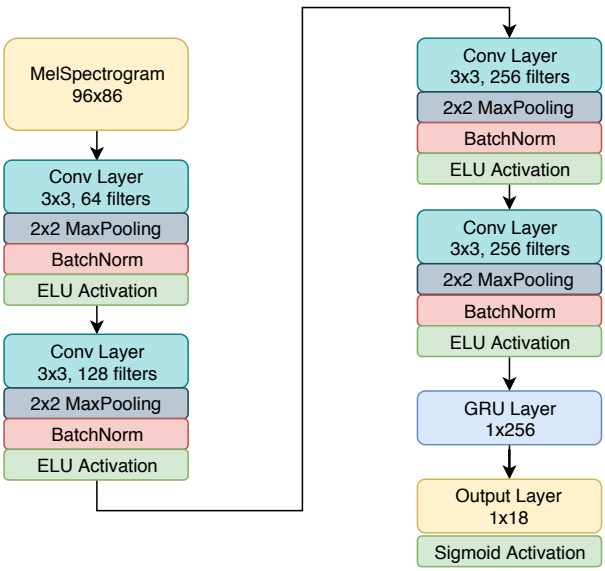


Figure 2.5: Gururani et al. [31] model architecture

set. MedleyDB contains 330 multi-tracks and Mixing Secrets contains 258 multi-tracks. In the work they considered only the 18 most present instruments. The clips in this case are *strongly* labelled, as they are annotated with a time resolution of 0.0464 seconds. Gururani et al. decided however to regroup the annotations by pooling the musical instruments activity over 1 second. The training split consists of 361 tracks and the testing split consists of 100 tracks. Moreover, data augmentation is employed: pitch-shifting was performed on the training set from 6 semitones lower to 5 semitones higher, with 1 semitone increments.

Their best performing model is a Convolutional Recurrent Neural Network, a model popular in music tagging and sound event detection. In these cases it is very useful to capture the temporal evolution of the input, as the wanted information is often contained in a small segment of a long temporal window. In our case the CRNN allows to better represent the temporal evolution of the frequency spectra. It is a Convolutional Neural Network with a recurrent layer in the place of a fully connected layer, which tracks the time evolution of the inputs.

In figure 2.5 we have a scheme of the architecture proposed. All audio is divided into 1 second inputs. Each snippet is then transformed in mel-spectrograms, with 96 mel-bands. The input then is fed through Convolutional Layers equipped with 3×3 shape filters and depth ranging from 64 to 256, followed by 2×2 Max-Pooling, Batch Normalization and ELU ac-

tivation. Finally, there is a Gated Recurrent Layer with 256 units, and then the Output Layer with 18 units and a Sigmoidal activation. They tested the CRNN on the class-averaged AUC and pooling results for a 10 seconds window.

2.4 Sound Event Detection

Sound Event Detection (SED) is a very popular task in the MIR research community, and it is closely related to Instrument Recognition. A *sound event* is an audio segment that a human would label as a distinctive sound impression in an acoustic signal. SED is employed to detect and classify *sound events*, in order to enhance multimedia indexing [65], scene recognition for mobile robots [11] and surveillance for a living environment [34]. Examples of sound events are gunshots, alarms, dog barks or human cries. Compared to Instrument Recognition, SED is a simpler task, as sound events of interest tend to be not correlated with the rest of the signal (differently from instruments playing along) and to have less harmonics of musical instruments.

Deep Learning approaches resulted in excellent results for this task as well. For example, Cakir et al. investigated both CNN and CRNN [9, 10] proving their effectiveness.

In order to help the SED research community to explore the AudioSet dataset [26], Google released a pre-trained model called VGGish [36]. The VGGish model can be used in two ways:

- *As a feature extractor*: the model converts audio input features directly into a semantically valuable, high-level 128-Dimensional embedding which can be fed as input to a classification model.
- *As part of a larger model*: use the model as a *warm start* for the lower layers of a model that takes audio features as input and adds more layers on top of the VGGish embedding. This can be used to fine-tune the starting model.

We employed the model as well in our research, as a feature extractor. Because of that, it is useful to review the model. The model takes its name from the classic VGG16 model [57]. The VGG16 architecture is shown in figure 2.6. Instead, the VGGish network is comprised of increasing depth using an architecture with small 3×3 convolution filters. The input size was changed to 96×64 for log mel spectrogram audio inputs. They used 4 groups of convolution/maxpool layers instead of five, and instead of a

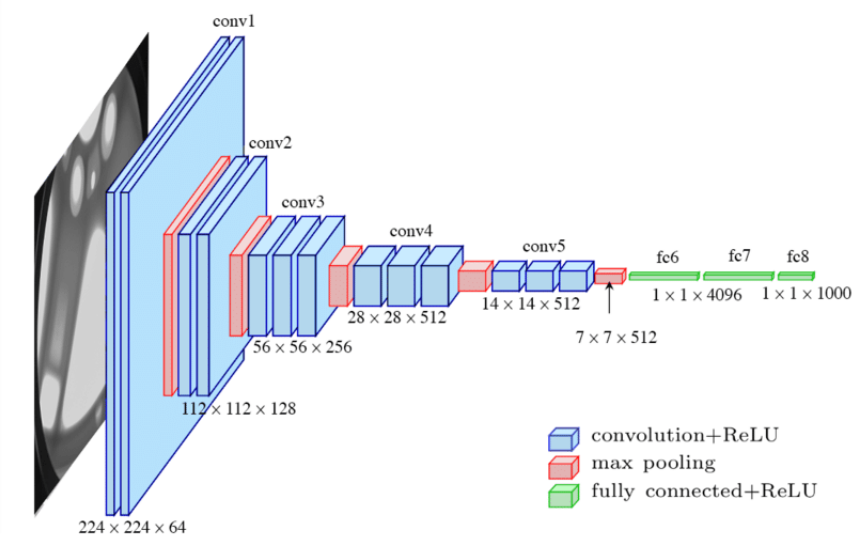


Figure 2.6: The original VGG16 architecture. Picture taken from Automatic localization of casting defects with convolutional neural networks, Ferguson et al. [22]

1000 units fully connected layer they used a 128 units fully connected layer, that acts as a Embedding layer. Moreover, each convolution block has one convolution operation less than the original architecture.

Before using the audio as input, it was down sampled to 16 kHz, STFT was computed with a window size of 25 ms and a hop size of 10 ms, and logarithmic mel-spectrogram with 64 bins is computed. Then, 0.96 seconds of audio as spectra is fed each time as input.

To summarize, it is possible to extract 128 embeddings, which are then post-processed with a PCA transformation, from each 0.96 seconds of audio, using a trained network on million of audio files extracted from YouTube. These are of course very powerful feature, as the model is trained on millions of clips.

Multiple Instance Learning

The advent of this pre-trained model and the proliferation of *weakly labelled* datasets, due to the lighter workload requested to human annotators, ignited the research of a solution for the Multiple Instance Learning (MIL) problem. Remember that *weakly labelling* means a clip is annotated as positive if at some point the target sound is present, and it does not matter if it is not present for the entirety of the clip. As we will work on *weakly labelled* data, it is important to review the most common techniques used in this framework.

Multiple instance learning is a variation on supervised learning, where each learning example contains a bag of instances. In MIL, a positive bag contains at least one positive instance. On the other hand, a negative bag contains no positive instances. Let us suppose we feed to the feature extractor an audio clip of s seconds, the results will be a feature vector of dimension $s \times 128$. We receive as output s independent predictions of detection from this model, but we need to pool them into one single prediction. The most common methodologies are:

- *Max Pooling*: final prediction only takes into account the highest prediction values among the instances.
- *Mean Pooling*: final prediction takes into account every prediction value among the instances, with the same weight.
- *Soft-max Pooling*: final prediction takes into account every prediction value among the instances, with weights increasing as the value of the prediction instance increases.

While investigating this problem, researchers discovered alternative solutions, such as the Attention Pooling or the Adaptive Pooling.

Kong et al. (2018) [43] experimented Attention Pooling within the AudioSet dataset and achieved remarkable results. Yu et al. (2018) [61] extended Kong's work and implemented Multi-Level Attention modules, to better exploit the advantages given by the attention method. Given these new pooling functions, Wang et al. (2018) [59] investigated how the different pooling methods affected the gradient of the loss function, possibly interfering with the optimal behaviour for the Neural Network.

Finally, McFee et al [51] developed a variation of the Soft-max Pooling Layer with extra weights, such that these parameters of the pooling function could be learned like a normal layer in a CNN, providing much more flexibility for the choice of the instances.

Chapter 3

Theoretical Background

In this chapter we review the theoretical background and the tools used in our work. In section 3.1 we make an overview of the physical properties of the musical instruments, and we describe the perceptual qualities for musical sounds. Then, in section 3.2, we focus on the signal processing tools used to inspect the audio and to extract relevant features for the acoustic properties, such as frequency spectra and Mel Frequency Cepstrum Coefficients. Finally, in section 3.3 we discuss machine learning techniques, with particular focus on Deep Neural Networks, Convolutional Neural Networks and the Multiple Instance Learning framework.

3.1 Properties of Musical Instruments

In order to perform automatic musical instrument recognition, it is necessary to understand the spectral features of the instruments, as described in section 2.3.3. To better understand their ability to characterize the sound of the instruments and how these spectral features are shaped, we discuss some modelling methods. Any musical instrument can be described as a vibrating system, set into excitation by an external force, and forced into resonance. The resulting mechanical oscillations are the source of the sound of the musical instruments. These systems are complex, governed by partial differential equations, therefore individual elements are not easy to separate in order to carefully detail every interaction of the system [23]. As we do not want to treat the model from the detailed perspective of the physics of the elements, but rather directly on the behaviour of the audio signal associated to a certain instrument, let us introduce some assumptions in order to discuss how the spectral features are originated from the vibrating system. Before discussing the interactions, we will give formal definitions of musical

concepts that will ease the discussion.

3.1.1 Pitch

The *pitch* is a perceptual property of sounds that allows their ordering on a frequency scale. It is the property which allows us to judge sounds as *higher* or *lower* with respect to each other.

In the frequency domain, *harmonic* sounds are described by a set of frequency components. The lowest partial component is the fundamental frequency, noted as F_0, and the other harmonics are quasi integer multiples of the fundamental. In a perfectly harmonic signal, the partials would be all formulated as kF_0, with k being ≥ 1 and integer. In quasi-harmonic signals such as the ones for pitched musical instruments, the partials follows a slight inharmonic pattern defined by the instruments. In the slightly inharmonic case, the partials can be formulated as:

$$f_k = kF_0\sqrt{(1 + k^2B)/(1 + B)}, \tag{3.1}$$

where f_k is the k-th partial and B is the inharmonicity coefficient, which varies depending on the instrument.

The pitch can be defined as the subjective impression of the fundamental frequency, not of the partials, of a particular sound.

3.1.2 Loudness

The *loudness* is the subjective perception of sound pressure, related to sound pressure level (SPL), frequency content and duration of a sound. The sensitivity of the auditory system changes as a function of frequency and of the SPL, and this relationship has been modelled by Fletcher and Munson, as shown in figure 3.1.

The threshold of audibility indicates the minimum sound pressure level perceivable by humans. The isophonic curves represent the needed SPL, for each frequency, to be perceived as the same loudness. They are expressed in Phons. The phon matches the sound pressure level in decibels of a similarly perceived 1 kHz pure tone. From the curves we can see that the ear performs best at a certain frequency range, between 2 kHz and 4 kHz. The bounds of the graphs show a minimum of 20 Hz and a maximum of approximatively 17k kHz.

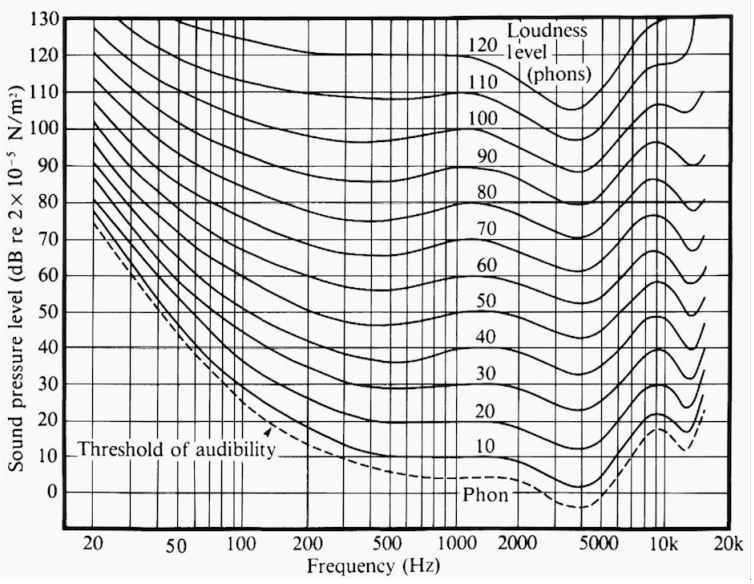


Figure 3.1: Fletcher-Munson curves, taken from the book The physics of musical instruments by Fletcher [23]

3.1.3 Timbre

The *timbre*, also known as tone colour or tone quality, is the perceived sound quality of a musical note, sound or tone. The timbre distinguishes between families of instruments such as string, wind or percussion instruments, but also enables listeners to distinguish different instrument in the same category, such as a guitar and a violin. From the Acoustical Society of America (ASA) Acoustical Terminology:

> *[The timbre] is that attribute of auditory sensation which enables a listener to judge that two non-identical sounds, similarly presented and having the same loudness and pitch, are dissimilar.*

It is closely related to both the frequency spectrum of the instrument and the temporal envelope of the sounds.

3.1.4 Acoustic Properties

As aforementioned, the problem of a sounding musical instrument can be divided in two major components: the imposing force, which set the fundamental oscillating frequency, and the resonator, set in motion by the force. As an example, in a violin the string is the exciter, and the wooden body is the resonator. The excitation combined with the vibrational modes of

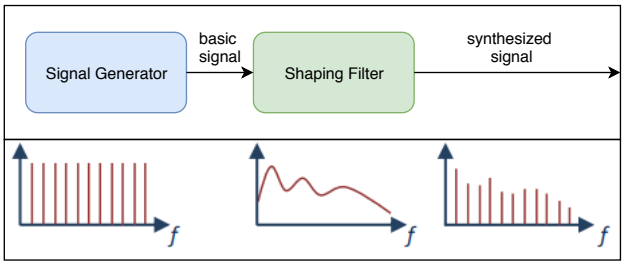


Figure 3.2: Source-Filter model

the resonator produce a sound pattern made up of individual components, named *partials*.

These partials tends to be located at quasi integer multiples of the fundamental frequency, which is the lowest one, if the signal is periodic. The resulting spectrum is *harmonic*, and is recognized by humans as the *pitch* property for the sound. On the contrary, percussive instrument are excited by aperiodic signal, like a tone burst, which results in a partials spectrum spread all over the frequency range, and are perceived as *unpitched* sounds.

This complex vibration pattern is then imposed to the resonator, which directly acts as a filter on the source signal and reshapes the amplitudes based on its frequency response. Since the frequency response is not dependent on the source but only on the geometry and materials of the resonance body, we can separate the filter effects and study which frequency region of the source are affected. These modulation are known as *formants*, and they are partly responsible on how we are able to distinguish the *timbres* of different instruments playing the same *pitch*. Let us summarize the interaction with a source-filter scheme, such as the one in figure 3.2

Moreover, we need to model the interaction over time. While it is true that the frequencies are related to the resonances, we do not know how the temporal envelope is defined. In order to study the temporal evolution, a sound is usually divided in three components: attack, sustain and release. Attack and release are present in all natural sounds, as they are a consequence of the vibrational excitation. The sustain, however, changes considerably among the instruments. For example, the piano and the guitar, which have struck/plucked strings as excitation, do not have a strong sustain, as their sounds rapidly begins to decay after the excitation. Blown instruments, however, have a characteristic sustain imposed by the player. If we study both the temporal envelope and the spectral envelope, we can model them together as a spectro-temporal envelope. Combining these two domains, we can finally visualize the complete informations our ears exploit

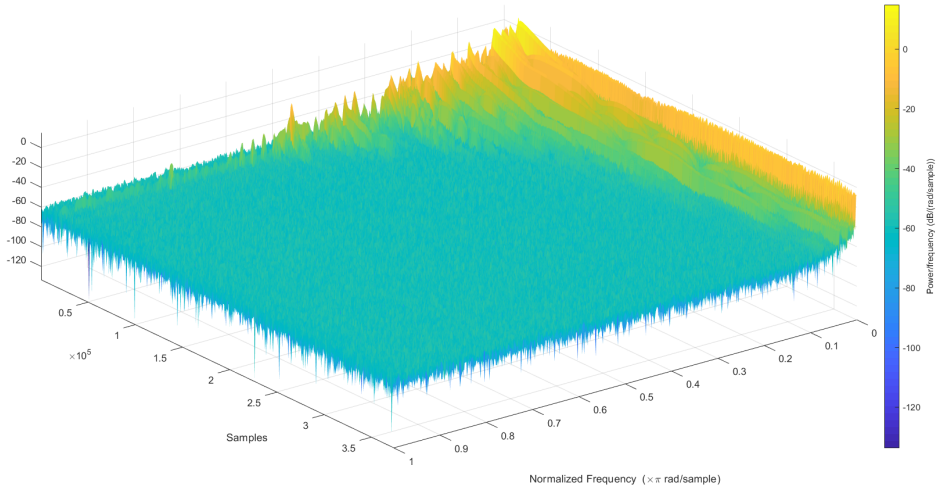


Figure 3.3: Spectrogram of a A3 grand piano note, where we can see the frequency decay over time (samples)

to differentiate timbres between instruments. This representation is very useful for our model as well, in order to learn to distinguish the various musical instruments, as each musical instrument has its own unique combination of the temporal envelope and the spectral envelope. In figure 3.3 we can observe the spectro-temporal evolution of a recorded A3 grand piano note.

3.2 Timbre characterization

In order to complete our task, it is fundamental to extract a meaningful representation from the audio signal, in order to provide an effective timbre characterization. We need to capture the characterizing details from the audio signal in order to easily represent the differences between the spectral features of the musical instruments, so that we can identify them. The audio signal can be either represented in the time domain or the frequency domain. In this section we review the frequency transformation (STFT) and how to shape the features in a scale which is closer to the human perception. These transformations are the most effective in order to obtain a time-frequency representation as described in the previous section.

3.2.1 Short Time Fourier Transform

To compute the frequency-domain representation of a discrete-time signal, the Discrete Fourier Transform (DFT) is commonly employed:

$$S(w_b) = \sum_{n_f=0}^{N_f-1} s(n_f)e^{-jw_b n_f}, \qquad b = -\frac{N_f}{2}, \ldots, \frac{N_f}{2}, \tag{3.2}$$

where:

- F_s is the sampling frequency,
- N_f is the number of samples of the total sequence,
- $s(n_f)$ is the n_f-th time sample,
- $w_b = 2\pi \cdot F_s \frac{b}{N_f}$ is the b-th frequency bin,
- $S(w_b)$ is the frequency content of the signal at the b-th bin.

The DFT however, gives us one frequency representation over the entire audio signal we use as input. That means, we only have one frequency frame. It becomes useless if we have a rapidly changing signal, which is the case in musical contexts. To overcome this limitation, the STFT is used.

The STFT is computed using a *windowing* process. The window is a simple function, that here we call $q(n_f)$, of length N_{win} which slides over the original signal with a hop-size of N_{hop}. The result of the windowing is the division of the original signal into a set of frames with length N_{win}, that are overlapped when $N_{hop} < N_{win}$. With the overlapping frames, we are able to get more precision in both frequency and time, defining the transformation as

$$S_r(w_b) = \sum_{n_f=0}^{N_{win}-1} s(n_f - rN_{hop})q(n_f)e^{-jw_b n_f}, \qquad b = -\frac{N_{win}}{2}, \ldots, \frac{N_{win}}{2}, \tag{3.3}$$

where:

- $s(n_f - rN_{hop})q(n_f)$ is the n_f-th sample of the r-th frame,
- $w_b = 2\pi \cdot F_s \frac{b}{N_f}$ is the b-th frequency bin,
- $S_r(w_b)$ is the r-th frame frequency content at the b-th bin.

Regarding the time-frequency precision, we must note that it is a trade-off. A longer frame size N_{win} corresponds to an higher frequency resolution and a lower temporal resolution. The frequency (Hz) resolution of a discrete-time signal is formulated as:

$$\Delta f = \frac{F_s}{N_{win}}. \tag{3.4}$$

To obtain the final representation of the *spectrogram*, the spectrum is computed for each frame and then they are concatenated in the time domain. As the phase of the spectra is not very informative for the musical instrument recognition task, we take only the magnitude of the frequencies. Moreover, as the spectrum of a real signal is symmetric, we can discard the negative bins and keep only the positive frequencies.

3.2.2 Mel-Spectrogram

As shown by psychoacoustics studies [58], the human auditory system does not perceive frequencies on a linear scale, rather on a logarithmic scale. Given this assumption, we need a representation of the signal that follows the pace. The solution to this problem is the Mel-Spectrogram, widely adopted by the MIR community.

The Mel-Spectrogram is obtained from filtering the standard spectrogram on the frequency axis. A particular set of triangular Mel-filterbank (figure 3.4) is applied, changing the linear scale to a logarithmic one. The conversion rate is defined as:

$$f_{log} = 2595 \log_{10}(1 + \frac{f_{lin}}{700}). \tag{3.5}$$

The Mel-filterbank is made of overlapping triangular filters, where each filters extract a single coefficient. It is needed to specify the number of filters, when creating the filterbank.

In figure 3.5 we can present examples of the representations described above, computed on an audio snippet of 10 seconds. The mel-spectrogram has been computed using a filter-bank with 12 bands.

3.3 Deep Learning Networks

In this section we revise the theoretical background on Deep Neural Networks (DNNs), providing the knowledge needed to understand the proposed method. Neural Networks [28] are models that aim to mimic the human

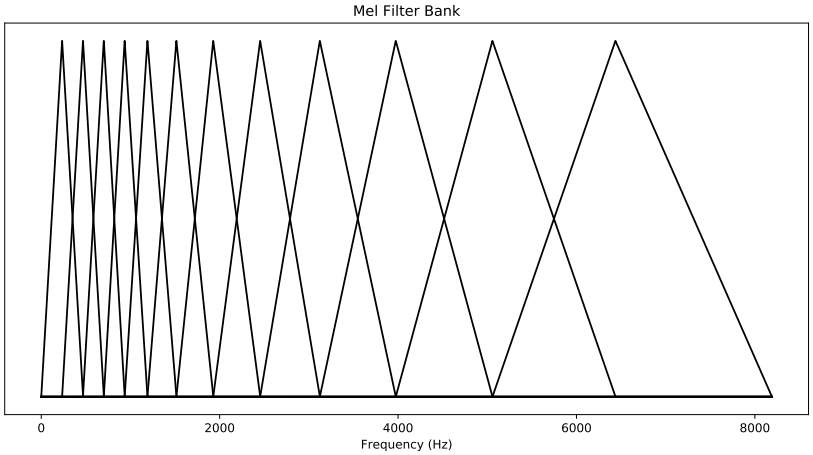
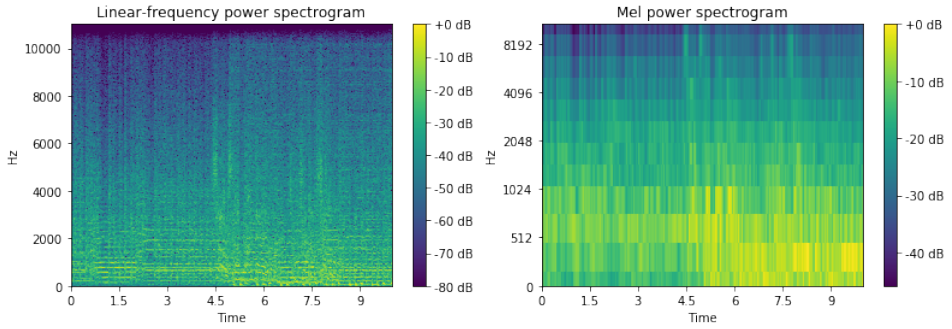


Figure 3.4: Mel-filterbank for 12 coefficients.



(a) STFT of the signal *(b) Mel-Spectrogram of the signal*

Figure 3.5: Spectral representation of the first 10 seconds of the song Fantasy Cops by Gee Tee. The change in musical instruments is visible at second 4 approximatively, changing from guitar only to guitar, synthesizer, bass and the lyrics "Just stop right there, you criminal scum".

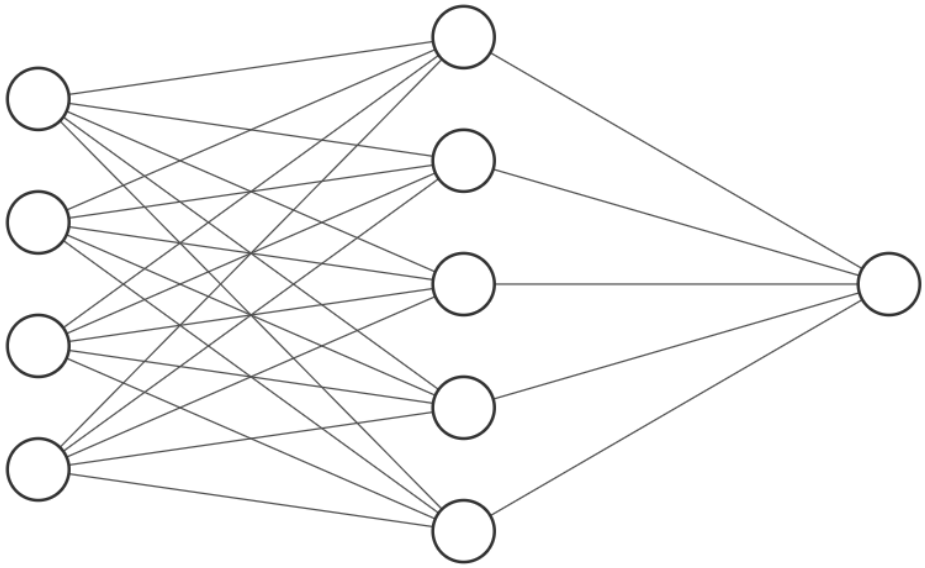


Figure 3.6: Example of a basic Neural Network architecture.

brain functioning, and its ability and flexibility towards learning. An example of Neural Network architecture is shown in figure 3.6. We can see how these model attempt to mimic the interconnected structure of the neurons in the brain, by having multiple computational nodes, called *neurons*, organized in sets, called *layers*. These layers are hierarchically stacked in order to extract a representation of the input data. Before using the network, we must *train* the model. This learning phase is referred to as *training*. In this phase, the network learns the representation of the input data. After the training phase, we can use the model to solve tasks on new, unseen samples.

Let us describe the basic components that constitute a Neural Network. The basic block is the *neuron*, shown in figure 3.7 (a). It is analogous to a biological neuron and represents a scalar value x. A *layer*, shown in figure 3.7 (b), is a set of nodes. Usually an *activation function* g, shown in figure 3.7 (c), is applied to each node of a layer, and nodes in a layer are independent from each other. These functions govern the activation of the neurons, often through non-linear relationships. When we connect together several input and output nodes, we have interconnecting layers, shown in figure 3.7 (d). If we have more layers than those two, the intermediary layers are called *hidden layers*, which are a step in the computational process which is not explicitly stated in the input-output relationship.

In fact, these models are called *deep* because we employ a multitude of stacked layers, which define the *depth* of the model. Similarly, the *width* is

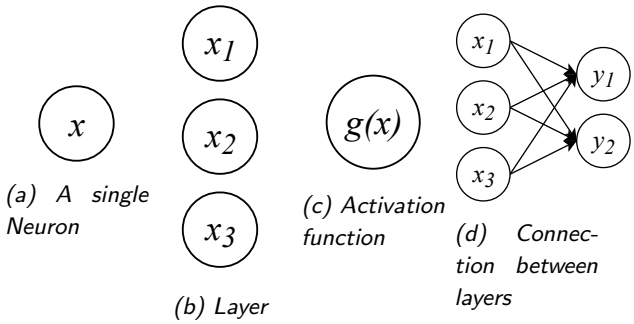


Figure 3.7: Graphical representation of the basic blocks for DNNs.

the number of nodes in one layer. In general, any Neural Network is used to perform a non-linear mapping between an input $\mathbf{x} \in \mathbf{X}$ and an output $\mathbf{y} \in \mathbf{Y}$. It can be formally defined as

$$\mathbf{y} = f(\mathbf{x}; \mathbf{w}), \tag{3.6}$$

where $\mathbf{w} \in \mathbf{W}$ are the *parameters* of the network. During the *training phase*, those parameters are iteratively adjusted in order to learn an optimal representation of the input, as we will describe in the next section.

3.3.1 Deep Neural Networks training

When approaching a problem with Deep Learning, the choice of the parameters which influence the training procedure is of fundamental importance, in order to achieve an adequate result. These type of parameters are usually referred as *hyperparameters*. Let us consider a Neural Network as the function $f : \mathbf{X} \to \mathbf{Y}$, where $\mathbf{X}$ is the input space and $\mathbf{Y}$ is the output space.

When training, the network is parametrized by its weights $\mathbf{w}$. The prediction $\hat{\mathbf{y}}$ of the network, which is the output of the network given by the input $\mathbf{x}$ and the weights $\mathbf{w}$, is compared to the *ground truth* value $\mathbf{y}$ and a *loss function*, called $J(\mathbf{w})$ can be computed. The loss function is a measure of effectiveness for the model.

In order to minimize the loss function $J(\mathbf{w})$, and achieve an effective result for the training phase, we need to iteratively adjust the weights $\mathbf{w}$. While training, we are searching for the optimal parameters $\mathbf{w}^*$ where the gradient of the loss function is 0:

$$\nabla J(\mathbf{w}^*) = \frac{\partial J(\mathbf{w}^*)}{\partial \mathbf{w}} = 0. \tag{3.7}$$

Since reaching this condition is not an easy task, the approach is to *iteratively* update the weights and get closer to the solution. One of the

most used optimization method is the gradient descent update, formulated as:

$$\mathbf{w}_{p+1} \leftarrow \mathbf{w}_p - \eta \nabla J(\mathbf{w}_p), \tag{3.8}$$

where, considering a training step p of the total procedure, η is the *learning rate*, $\mathbf{w}_{p+1}$ are the adjusted weights, and $\nabla J(\mathbf{w}_p)$ is the gradient for the weights $\mathbf{w}_p$, computed as $\partial J(\mathbf{w}_p)/\partial \mathbf{w}$. The *learning rate* η can be adaptively controlled, and highly influences the convergence of the algorithm, for both the computational time and the overall result. It should decrease as it approaches the local minima. One of the most popular optimizer for adaptive learning is Adam [41]. When the update of the weights $\mathbf{w}_i$ is computed, we need to update every parameter that led to this prediction. This process is known as *backpropagation*, which consists in updating each weight in the network starting from the error on the final output. In fact, we can compute the weight update for each node in each layer, using the *chain rule* of calculus on the gradient. The chain rule is used to compute the derivatives of functions created by composing other functions whose derivatives are known. This way, the error takes into account every stage of the network and adapts the weights accordingly. The weight update for a generic parameter $w_{i,j}$ between node i and node j is computed as:

$$\frac{\partial J}{\partial w_{i,j}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{i,j}}, \tag{3.9}$$

where $w_{i,j}$ is the parameter we want to update between node i and node j, y_j is the output of the node j, and x_j is the input of the node j.

Activation functions are another important element in DNNs. They control the activation of the neurons and introduce non-linearity in the system, allowing to learn more complex relationships. The introduction of Rectified Linear Unit (ReLU) [27], for example, enhanced the computational speed for many Neural Network applications and helped to solve the vanishing gradient problem. This difficulty occurs when the gradient $\nabla J(\mathbf{w})$ becomes very small and it blocks the training procedure of the network. The ReLU activation is formulated as:

$$g(\mathbf{x}) = max(0, \mathbf{x}). \tag{3.10}$$

As we said, it is widely used to control the *activation* of neurons. It is computational cheap, does not saturate and it is sparsely activated, which is often useful to avoid the *overfitting* of a model.

When training a Neural Network, the dataset is usually divided in 3 parts: the *training* data, the *validation* data and the *testing* data. The

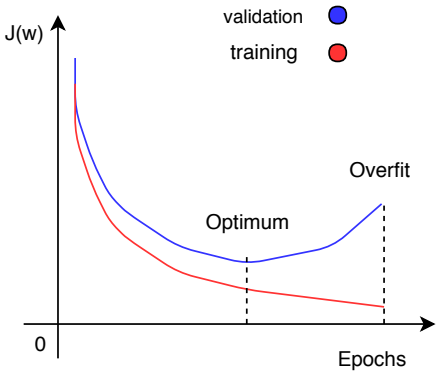


Figure 3.8: Overfitting behaviour during the learning phase

machine is trained on the first set, periodically evaluated on the second set, and at the end of the training procedure, finally tested on the third set. The testing data is never shown to the model during the training procedure, where we alternate instead the usage of the training data to update the weights of the model, and the validation data that is used to assess the performances. The model is evaluated on the validation set at the end of each *epoch*, which is a full iteration on the training data. In this way, we are able to spot when the model begins to lose the generalization of the classes distribution. This is called *overfitting*, and it is graphically represented in figure 3.8. One simple way to counteract this phenomenon is to use *early stopping*, where we halt the training procedure when the validation loss stops to decrease, and the training loss continues to decrease towards convergence. Other common approaches are to employ a *dropout* layer, which randomly blocks nodes in the net to avoid an instant overfitting of the architecture, and the usage of *regularization*, which does not allow the weights to grow excessively, hiding the effects of the other weights.

Among the various activation functions, it should be noted that, in case of a multi-class and single-label classification the output layer should have a *soft-max* activation, while in the case of multi-class multi-label a *sigmoid* activation should be employed. The advantage here lies in the fact that these activations output values in the probability range, which the domain we need for the output, comprised of label probabilities. The difference between those two functions is that the *soft-max* distributes the probability throughout each output node, so that the most probable class will have the higher probability, and all the output values will sum to 1. Instead, with the *sigmoidal* function, each probability class is independent from the other, so that every class is disjoint from the other probabilities. This way, the sum

of all the probabilities can surpass 1.

Having discussed the basics of the behaviour of these nets, we introduce some of the most common layers used in these architectures.

3.3.2 Feedforward Neural Networks

Dense layers are the most basilar layer configuration used in DNNs. Their name derives from the complete connectivity of the neurons between the layers. A network comprised only of dense layers is called a *feedforward neural network*. Dense layers are formulated as:

$$\mathbf{y} = g(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}), \tag{3.11}$$

where $\mathbf{x} \in \mathbf{X}$ is the input, $\mathbf{b}$ is the biases vector, $\mathbf{y} \in \mathbf{Y}$ is the output and $\mathbf{W}$ is the weight matrix. As discussed before, g is the non-linear activation function. The dense layer serves as a mapping between the input space $\mathbf{X}$ and the output space $\mathbf{Y}$. These unidirectionally oriented layers are used in *feedforward neural networks*. A network is *fully connected* when each neuron receives an input from each neuron of the previous layer, and computes the output with equation 3.11. In figure 3.9 we can observe a graphical example of a classic feedforward architecture. While being the most straightforward neural network architecture, it is a perfect example to show how the neurons, which are simply connected between each other with the weight matrix $\mathbf{W}$, can build a powerful representation. However, dense layers do not scale well with input like images, or spectrograms. For example, a 200×200 image would lead to a dense layer with $200 \times 200 \times 3 = 120000$ weights, with 3 being the number of colour channels. As we add other layers, the number of parameters exponentially grows, and the network is at high risk of overfitting. To overcome this problem, researchers developed Convolutional Neural Networks.

3.3.3 Convolutional Neural Networks

Convolutional Neural Networks demonstrated impressive results with computer vision tasks such as image classification [45] or object recognition [49]. They are a specialized kind of neural network, with the aim of processing data that has a grid-like topology. Such inputs are time-series data, which can be thought as 1-Dimensional grids, images, which are 2-Dimensional grids of pixels, and spectrograms, which can be interpreted as images with axis in 2 different domains, time and frequency. The name *Convolutional*

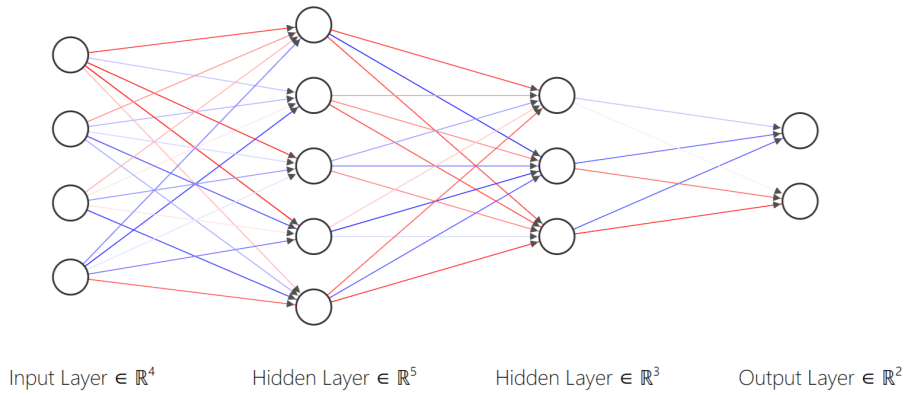


Figure 3.9: Neural Network architecture using only dense layers. As we can see, the layers are fully connected.

Neural Networks implies that the network employs a mathematical operation called *convolution* in its layers, in place of general matrix multiplication. Given a time series $v(t)$, with both v and t real-valued, if we want to make a weighted average with a weighting function $l(a)$ at every step a, we can use the convolution operation, which is described as:

$$m(t) = \int v(a)l(t-a)da \qquad \text{or} \qquad m(t) = (v * l)(t), \tag{3.12}$$

where m is also called *feature map*, v is the input and l is the *kernel* of the convolution. However, when working with digital signals, we are in a domain where time steps are discretized. Therefore, if we have values at regular intervals, the discrete convolution can be formulated as:

$$m(t) = (v * l)(t) = \sum_{a=-\infty}^{\infty} v(a)l(t-a). \tag{3.13}$$

Of course, as we often find ourselves using convolutions on 2-Dimensional inputs such as images, the operation must be brought in its multi-dimensional form. Given a 2D input named V and, therefore, a 2D kernel L, we can define the 2 dimensional convolution M as:

$$M(i_t, j_t) = (V * L)(i_t, j_t) = \sum_{m_l} \sum_{n_l} V(m_l, n_l)L(i_t - m_l, j_t - n_l), \tag{3.14}$$

where (i_t, j_t) are the coordinates of the input and (m_l, n_l) are the coordinates of the kernel function. Given these assumptions, we can now incorpo-

rate the *convolution* in one of our layers. The operation in a convolutional layer can be formulated as:

$$\mathbf{y}^c = g(\sum_{r=0}^{R-1} \mathbf{W}^{cr} * \mathbf{x}^r + \mathbf{b}^c), \tag{3.15}$$

where $\mathbf{y}^c$ is the c-th channel output, $\mathbf{x}^r$ is r-th input channel, $*$ is the convolution operation, $\mathbf{W}^{cr}$ is the convolution kernel that associates the r-th input channel and c-th output channel, $\mathbf{b}^c$ is the c-th bias vector for the output channel, and g is the activation function. A 2D convolutional kernel sweeps over an input and the weights that compose the kernel are applied to the input area. This is called *parameter sharing*. It results in vastly reducing the number of trainable parameters, and the time needed for the model to complete the training phase.

Another property of convolutional layers is the *local connectivity*. As a result of the filtering, the convolutional layer outputs a representation of the local activations of the input. The result is the local correlation between kernel and input, and it is an excellent way to reduce dimensionality. During the training phase, the kernels can learn patterns which reduce the loss. Stacking multiple layers together, we are able to learn complex representations based from the patterns of the previous layer.

Another layer commonly used in CNNs is the *pooling layer*. It is usually employed after the convolutional layer, and performs a dimensionality reduction of the input, with a summary statistic of the nearby inputs. Two typical pooling functions are the *max-pooling*, which computes a maximum value of a given patch, and the *average pooling*, which computes the average of a given patch. In both cases, pooling helps to render the representation *invariant* to small translations of the input. This is a very useful property when the presence of a pattern is more important than where it actually takes place. For example, in case of face recognition, we do not need to know where the eyes are exactly: we just need to know that there is one in the right side of the face, and one in the left side of the face.

Summarizing, the typical Convolutional Neural Networks is comprised of a repeating block of convolutional layers and max-pooling layers with an activation functions such as ReLU, and then one or more fully connected layers, which utilize the reduced input to perform a non-linear mapping and extract the output. In the case of classification, the output layer is usually the probability value of belonging to each class of the model. In figure 3.10 we can observe a graphical example of a classic CNN architecture.

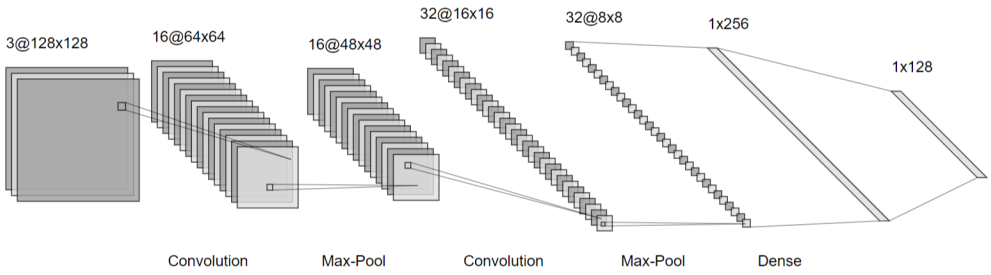


Figure 3.10: Typical architecture of a CNN, with convolutional, max-pooling, and dense layers.

3.3.4 Transfer Learning

Transfer Learning [28], or *domain adaptation*, refers to the technique of generalizing what a model has learned in one domain (distribution P_1) to another domain (distribution P_2). We assume that the representation learnt for modelling distribution P_1 are relevant for describing the variations in distribution P_2. This could be true in a supervised learning context, where the input may be the same for the two distributions but the target may be of different nature. For example, in image recognition, if a model learns to recognize edges and visual shapes, geometric translations and changes of light, a model which is able to recognize *dogs* might be easily translated to a model which efficiently recognizes *cats*. In our work we employ *transfer learning*, as we describe in chapter 4, in order to exploit the feature extractor process of another model which has been trained on a very large dataset. We investigate if it possible to adapt the knowledge of the VGGish model [36] trained by Google for Sound Event Detection, to the domain of our task, which is Instrument Recognition.

We have several choices when performing transfer learning. We can take a trained model, and use its first n layers to train a network for our task. In this way, we start from the weights of these layers, which already learnt a meaningful representation, and we can either *fine tune* the entirety of the first network or we can use new layers on top of the first n to model our new distribution. In both cases, the first n layers are in each case fine tuned. When doing so, a small learning rate should be employed to avoid the distortion of the weights.

Another approach is to take the first n layers of the trained model and use them as a feature extractor, without any fine tuning stage. In this case,

usually a shallow classifier is used on top of the extracted features. If the layers extract meaningful representations, this architecture results powerful and fast to compute.

3.3.5 Multiple Instance Learning (MIL)

In the context of Sound Event Detection, systems are typically trained with supervised learning approaches. These models are used to map an audio signal to a sequence of events, taking place in the audio excerpt. Ideally, as these models should produce time-varying labels for each instant of a recording, training from *strongly* labelled data would be optimal. However, these kind of labels are very costly to acquire. As a consequence, it is often preferred to collect data which is *weakly* labelled, due to the lower annotation costs. Moreover, they are annotated at a coarse time resolution, for example 10 seconds clip. To fully exploit the data, dynamic predictions are *pooled* across time to form static predictions. This is a common framework for SED and it is called *Multiple Instance Learning* [51]. In our work, the approach relies on *weakly labelled* data only, so it is fundamental to review the theoretical background for the approaches employed in this framework. In MIL we do not know the label values of every time instance for the training clips. Instead, the instances are grouped into *bags*, and only the label of the bag is known. The labels obey the *standard multiple instance (SMI) assumption*. A bag label is positive if and only if the bag contains at least one positive instance. A neural network predicts the probability value for each class, which is a independent binary classification problem, and then a pooling function aggregates the *frame level* probabilities into a *clip level* probability. This probability can be the confronted with the label value, and the loss of the network can be computed. In figure 3.11 we graphically represent how the most common pooling functions work.

Several pooling functions are commonly used in the MIL framework. Let n (or m) be the number of instances in the clip, $\hat{y}_n \in [0, 1]$ be the predicted probability for a class at the n-th frame, w_n be their associated weights and $\hat{y} \in [0, 1]$ be the aggregated clip-level probability for the same class. Moreover, let $h \in [0, 1]$ be the *ground-truth* label of the clip. The loss function usually employed is the *cross-entropy*:

$$J = -h \log \hat{y} - (1 - h) \log(1 - \hat{y}). \tag{3.16}$$

We can decompose its gradient with respect to the frame level probabil-

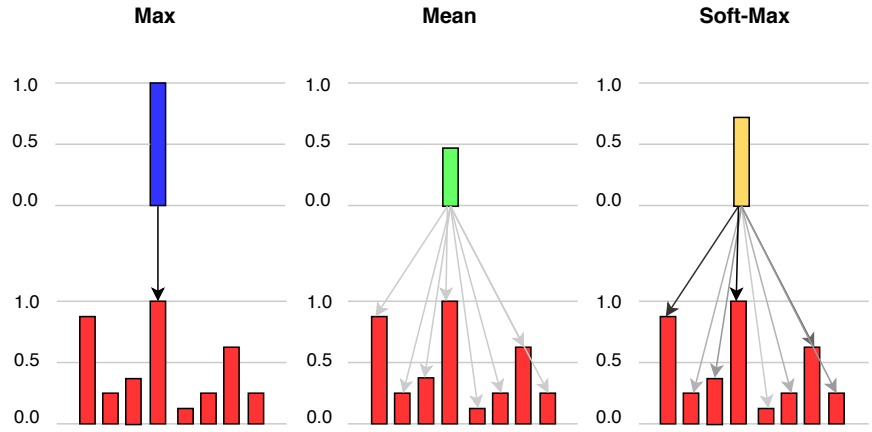


Figure 3.11: Effect of different pooling policies on a bag of instances. Here we can see how all the predictions in a bag are pooled into one single prediction. The arrow colour opacity is proportional to the weight magnitude.

ities $\hat{y_n}$ (and, if needed, the frame level weights w_n) using the chain rule:

$$\frac{\partial J}{\partial \hat{y}_n} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \hat{y}_n}, \qquad \frac{\partial J}{\partial w_n} = \frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_n}, \tag{3.17}$$

where the first term

$$\frac{\partial J}{\partial \hat{y}} = -\frac{h}{\hat{y}} + \frac{1-h}{1-\hat{y}} \tag{3.18}$$

does not depend on the pooling function. The second term, $\partial \hat{y}/\partial \hat{y}_n$ and $\partial \hat{y}/\partial w_n$ is calculated for each pooling function.

Now let us discuss the different pooling functions in detail. The max-pooling function simply takes the largest probability among $\hat{y}_n$ to be y, and it is formulated as:

$$\hat{y} = \max_n \hat{y}_n. \tag{3.19}$$

The max-pooling function effectively propagates the error to only the maximum value, since its gradient is

$$\frac{\partial \hat{y}}{\partial \hat{y}_n} = \begin{cases} 1, & \text{if} \quad \hat{y}_n = \hat{y} \\ 0, & \text{otherwise} \end{cases}, \tag{3.20}$$

which means that only one instance receives a non-zero gradient. As a consequence, if a event occurs multiple times during a clip, the occurrences in the non maximum frame may be easily missed, and potentially be learned

as a false negative. Another common solution is the average pooling. With average pooling, each instance is considered with the same weight:

$$\hat{y} = \frac{1}{n} \sum_n \hat{y}_n. \tag{3.21}$$

In this case the gradient is constant regardless of the considered instance:

$$\frac{\partial \hat{y}}{\partial \hat{y}_n} = \frac{1}{n}, \tag{3.22}$$

it is evenly distributed between all the instances, which implies that in a negative bag, every instance probability will decrease, while in a positive bag, each instance probability will increase. This can be harmful, in case only one instance is positive, and may cause many false positives.

The exponential softmax pooling computes y as a weighted average of y_n, assigning larger weights to larger values. The pooling function is formulated as:

$$\hat{y} = \frac{\sum_n \hat{y}_n \, \exp(\hat{y}_n)}{\sum_n \exp(\hat{y}_n)}, \tag{3.23}$$

and its gradient is formulated as:

$$\frac{\partial \hat{y}}{\partial \hat{y}_n} = (1 - \hat{y} + \hat{y}_n) \cdot \frac{\exp(\hat{y}_n)}{\sum_m \exp(\hat{y}_m)}. \tag{3.24}$$

the gradient is always positive and suffers of the same drawbacks of the average pooling. The tendency of causing too many false positives however is mitigated by the numerical value of the gradient, which is smaller with respect to the average pooling one.

Finally, in the attention pooling, the weights w_n for each instance are learned in the network as a layer. The clip level probability is then computed as a weighted average:

$$\hat{y} = \frac{\sum_n \hat{y}_n w_n}{\sum_n w_n}. \tag{3.25}$$

This function appears to be flexible and is widely adopted by researchers, with variants such as the multi-attention model [61], where the attention module is used multiple times between fully connected layers of the network and then these outputs are concatenated together. The gradient of the loss must be computed with respect to y_n and the weights w_n, and is formulated as:

$$\frac{\partial \hat{y}}{\partial \hat{y}_n} = \frac{w_n}{\sum_m w_m}, \qquad \frac{\partial \hat{y}}{\partial w_n} = \frac{\hat{y}_n - \hat{y}}{\sum_m w_m}. \tag{3.26}$$

When the value $\hat{y}$ is positive and the probability $\hat{y}_n$ is large, a large weight w_n will be learned, while if the probability $\hat{y}_n$ is small, a small weight

w_n will be assigned to the instance. This is the intended behaviour, as it facilitates to select the frames according to their probabilities. However, when the value $\hat{y}$ is negative, the opposite phenomenon will occur: due to the formulation of the gradient with respect to w_n, larger weights w_n will be assigned to small probabilities $\hat{y}_n$, and smaller weights will be assigned to larger $\hat{y}_n$ probabilities. This behaviour will boost the weight of small positive probabilities, which will be incorrectly highly considered. As a consequence, the model may classify many samples as false positives.

In our work, we use *weakly* labelled data in the MIL framework described above. It is fundamental to check how the gradient of the pooling functions behaves, as it will directly affect our results. As we must tradeoff the performances of the classification task when choosing the pooling functions, we decided to employ a function described by McFee et al. [51] as *adaptive softmax pooling*, or *autopooling*. This function behaves the same way as the exponential softmax pooling, but with weights $\alpha \in \mathbb{R}$ which can be learned along with the optimization of the network. It can be formulated as:

$$\hat{y} = \frac{\sum_n \hat{y}_n \exp(\alpha \hat{y}_n)}{\sum_n \exp(\alpha \hat{y}_n)}. \tag{3.27}$$

Since α is a parameter, the function is able to interpolate between the behaviour of different pooling functions. For example, when $\alpha \to \infty$, the function approaches the max pooling operator, and when $\alpha = 0$, the equation reduces to the mean pooling operator. When $\alpha = 1$, we are again in the case of the standard softmax pooling operator.

Chapter 4

Method Overview

In this chapter we describe the principles of our method and the basic architecture of the model. Our goal is to build a model able to identify musical instruments in audio clips. The methodology, introduced in chapter 4.1, can be divided in two phases. In 4.1.1 we discuss the extraction of features, and in 4.1.2 we discuss the architecture for the classifier.

4.1 Method

In our work, we approach the problem of recognizing each active instrument in each clip of a labelled dataset. Each musical instrument is a *class*. We approach a *multi-class multi-label* problem, where each music excerpt is classified for N_M classes, and all of them can occur at the same time in the music excerpt. Hence, we work in a polyphonic context. Our methodology employs Neural Networks and *transfer learning*. Transfer learning refers to take advantage of the knowledge learned in a domain, in a new related domain. This process is effective only if the features learned for the first task are general [60] and relevant for the second task, as described in section 3.3.4. In our work we explore how the features learned for the task of Sound Event Detection (SED) perform in musical instrument recognition. In SED it is usually easier to extract meaningful information about the frequency content when compared to musical instrument recognition, due to the harmonic content of the audio excerpts being generally more distinct. Nonetheless, the semantic representation provided by models trained on large datasets for SED can be very valuable also in our task. In figure 4.1 we provide a general schema of our model. Starting from raw audio, we extract features using the VGGish model released by Google [36], and then we feed the features into the classifier, which will be trained in a supervised

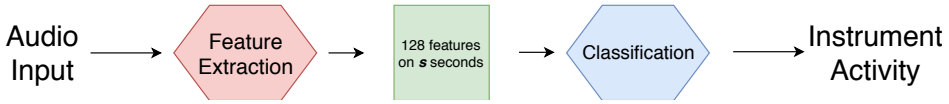


Figure 4.1: Overview of pipeline for our method.

learning fashion. In this chapter we will not specify the number N_M of musical instruments for the dataset nor the length N_S in seconds for the temporal resolution. These variables will be defined for each investigated dataset later in chapter 5.

4.1.1 Feature Extraction

In our model, we exploit the VGGish model, as it is trained on a large dataset by Google for the SED task. We investigate if it also performs consistently on the musical instrument recognition task. The VGGish model is a convolutional neural network. We employ it as a *feature extractor*. We take the model as it is trained, and we do not update its weights. In this way, we maintain the learned knowledge from the previous task, and we train our classifier on top on this model. The scheme of the model is presented in figure 4.2. The audio input is divided into 1 second chunks, hence this is the minimum temporal resolution for our model. After downsampling the audio to 16 kHz, the mel-spectrogram transformation is applied, and the resulting input is a matrix with dimensions 96 (time frames, from windows of 10 ms) × 64 (frequency bins). This spectrogram is passed as input to the convolutional neural network, which is composed of different blocks. We divided the blocks into 3 categories: C1, C2 and FC. The model has 2 C1 blocks, then 2 C2 blocks, and finally 3 FC.

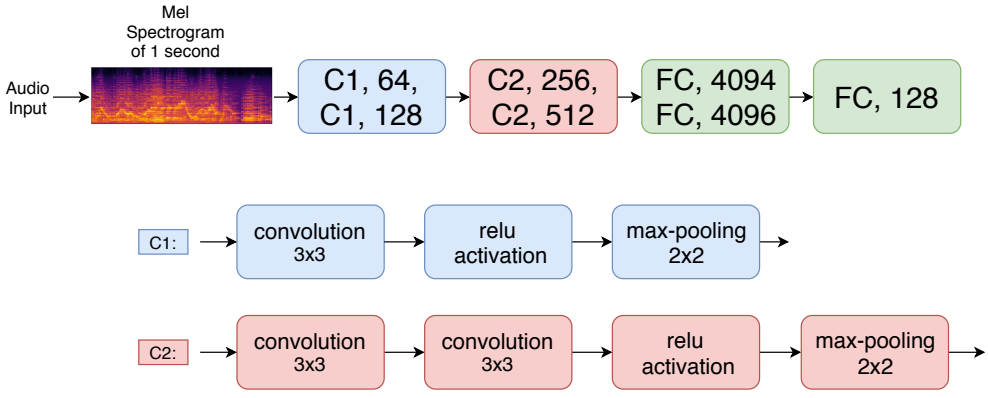


Figure 4.2: Scheme of the VGGish model developed for AudioSet, with each block explained.

Hyperparameters				
Layer	n. of Conv.	Filters	Kernel size	Pooling Size
C1.1	1	64	3×3	2×2
C1.2	1	128	3×3	2×2
C2.1	2	256	3×3	2×2
C2.2	2	512	3×3	2×2

Table 4.1: Hyperparameters for the VGGish model.

The C1 block is constituted as:

- A two dimensional convolutional layer.
- A ReLU activation function, employed to perform a non-linear mapping and to avoid the evanescent gradient problem.
- A MAX pooling layer, which performs down-sampling.

After those two blocks, the C2 series begin. These kind of blocks are similar to the C1 blocks, with the exception that they have 2 convolutional layers instead of 1. CNNs use additional hyperparameters which are not present in standard *feedforward networks*:

- The number of filters, which is the number of feature maps produced by the convolution layer.
- The *kernel size*, which is the dimension of the filter used for convolution.
- The *pooling size*, which is the dimension of the area considered for subsampling.

These parameters are specified in table 4.1. After the two convolutional blocks, the network implements the three *fully connected* (FC) layers. The first two layers have 4096 hidden units, where the output of the convolutional section is used as input. Then we apply a final dense layer, which has 128 hidden units.

These 128-Dimensional features are the final product of our feature extractor. They are a meaningful representation of the signal's characteristics, and in order to provide a more orthogonal representation with respect to those 128 dimensions, we apply a *rotation* of the axis through a Principal Component Analysis operation, maintaining the same number of dimensions. In fact, PCA is usually employed to *reduce* the dimensionality of a

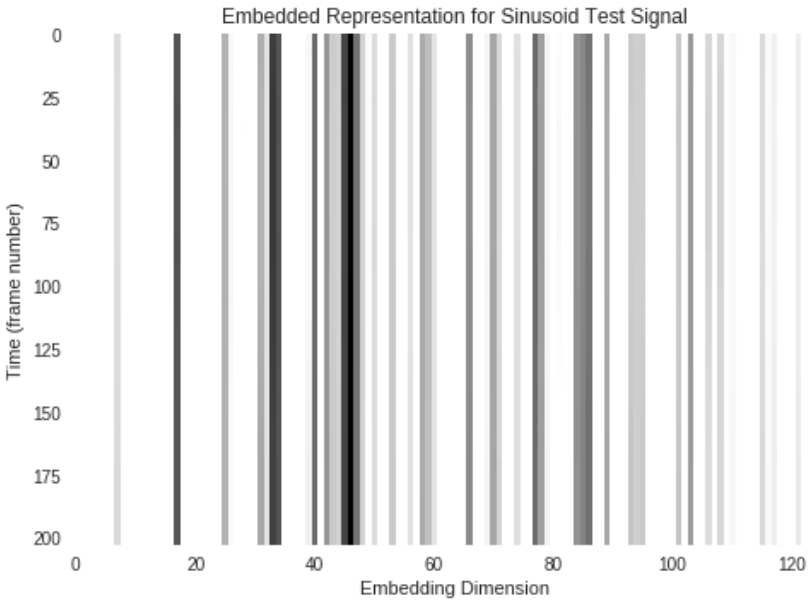


Figure 4.3: Features extracted from the VGGish model for a test sine wave. We can observe how the periodicity of the signal is recognized as the features are stationary.

feature space, by finding the directions with more variance. In our case, we find the directions, but we do not reduce the number of dimensions, which means we *rotate* each axis towards the direction with more variance. This helps us to obtain a more orthogonal representation. In figure 4.3 we can observe the feature representation of a test sine wave, for over 200 seconds of audio. We can see how the representation is scattered in different dimensions, and how these extracted features are *stationary* during the clip length, due to the periodicity of the signal.

The extraction of these features is computationally expensive: we need to load the audio, apply the mel-spectrogram transformation, create the instance of the model and extract the feature vector. However, it can be easily done only once as a pre-processing for the dataset. This becomes an advantage, as we can build a shallow model for classification which leads to fast training time. We can properly explore different classification configurations, without allocating an excessive amount of time waiting for the training procedure to halt. Often, due to the computational cost, training a Deep Learning model can take hours if not days, and this long waiting times can damage the research development. Using those features however, the largest portion of computational cost is met during pre-processing, and we can hastily make several experiments for the classifier architecture.

4.1.2 Classification

Given the input, which is an audio clip, we want to output the *class probability* for each instrument. The *class probability* is the probability of a clip to belong to a class assigned by the model. In this way, our model attempts to recognize if the instrument is present or not in a music excerpt, and with the probability of its presence. As we already extracted semantic features from the previous step using VGGish, we employ a different model for the final classification. We use *feedforward* networks, which define a mapping $f : \mathbf{X} \rightarrow \mathbf{Y}$, where $\mathbf{X}$ is the input space and $\mathbf{Y}$ is the output space, such as:

$$\mathbf{y} = f(\mathbf{x}; \mathbf{w}), \tag{4.1}$$

where $\mathbf{w} \in \mathbf{W}$ are the parameters of the model. The objective of the training procedure is to minimize the loss function. Given the loss $J(\mathbf{w})$, we can define the objective function of the network as:

$$\min_{\mathbf{w}} J(\mathbf{w}). \tag{4.2}$$

The loss is computed from the output labels of the dataset. We can define the output of our model as a vector $\hat{\mathbf{p}}$, comprised of N_M *class probabilities*. We threshold it in order to have a decision value, which is *present* or *absent*, and compare it to the *ground truth* of the label in the dataset. During the training procedure, the model updates its parameters $\mathbf{w}$ in order to minimize the loss function, in a iterative backpropagation fashion.

Our model relies on a ensemble of *feedforward network* for the classification section. The general architecture employs as inputs the features described in section 4.1.1, resulting in a matrix of dimensions $N_F \times N_S$, where $N_F = 128$ and N_S is the temporal length in seconds of the chosen audio clips. We exploit the power of dense layers by separating each time frame and parallelizing the classification. For each input of 128-Dimensional vector, we have a *feedforward network* which will be finally concatenated with the others through the means of a pooling function. A schema of this architecture is shown in figure 4.4. In fact, our architecture is *distributed*, or parallelized, for frames $s_i \in S$, where each frame is 1 second long.

For each frame s_i, we take its corresponding 128-Dimensional features, and we feed it to a feedforward network, with 3 blocks. The first 2 blocks are organized in the same way, constituted of:

- A fully connected layer of d_i hidden units.

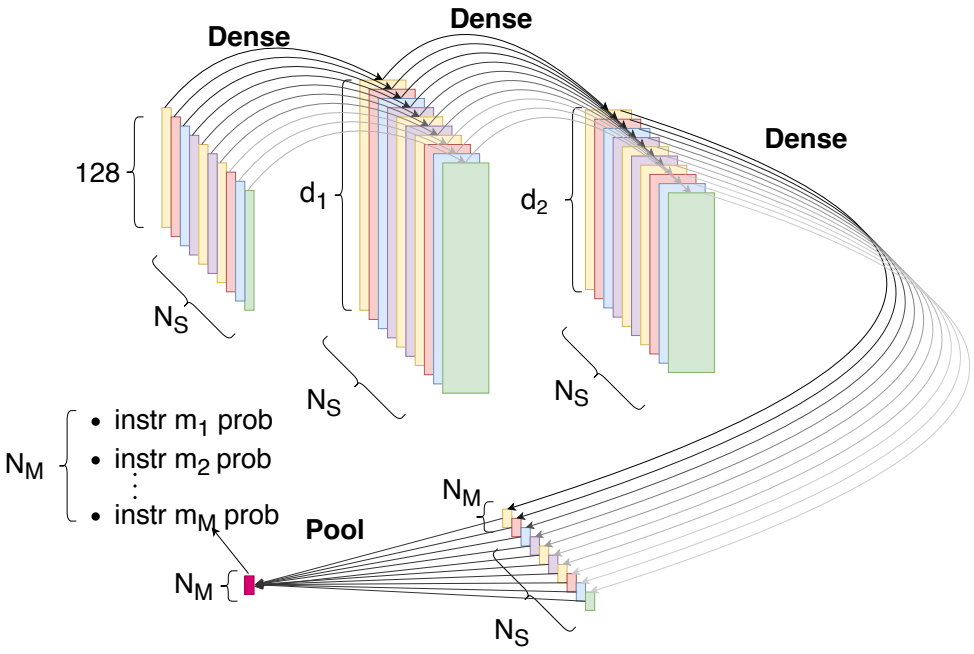


Figure 4.4: Architecture of the classifier, starting from the VGGish features. We can see the parallelization of layers which entails its time-distributed architecture, which finally converges to a single prediction per class through the pooling layer.

- A *batch normalization* layer used to perform normalization on the feature maps. It is used to smooth the gradient function and achieve faster convergence.
- A ReLU activation function, employed to perform a non-linear mapping and to avoid the evanescent gradient problem.
- A dropout layer, which randomly *drops* a percentage of the units in the layer in order to avoid the overfitting of the network.

These two block are used to perform an expansion of the dimensionality on the original feature vector. The number of hidden units d_i, with i integer $\in [1, 2]$, varies in the experiments, but it is constant regarding the frames s_i, and defines the *width* of the fully connected layers. In figure 4.5 we graphically summarized the composition of these blocks.

After these 2 blocks, we have a fully connected layer with N_M hidden units. This layer is used to make predictions for each one of the $m_i \in M$ instruments. For the activation function of this layer, we can choose between the *softmax* function and the *sigmoid* function. These are used in classification problems to transform the outputs of the layer into probability

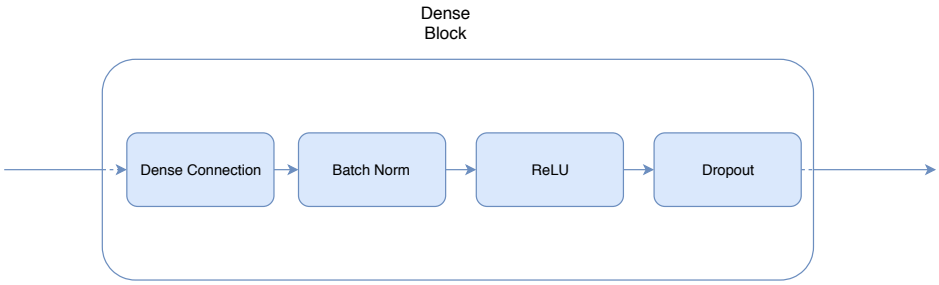


Figure 4.5: These are the detailed components for the first two dense layer in our architecture.

ranges (i.e. $\mathbf{y} \in [0, 1]$). The choice of this function must be done accordingly with the labelling of the training set. For example, if we have a training set labelled only with one instrument per clip (a *predominant instrument* setting), we should use a *softmax* activation function. The motivation of this choice is that the softmax function distributes the probability among all the nodes, such that the sum of every probability must be 1. The standard softmax activation function $g : \mathbb{R}^{N_Z} \to \mathbb{R}^{N_Z}$ is defined as:

$$g(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{i=1}^{N_Z} e^{z_i}} \qquad \text{for } j = 1, \ldots, N_Z, \tag{4.3}$$

where $\mathbf{z}$ is the input vector and N_Z is the dimensionality of $\mathbf{z}$. However, in a purely polyphonic setting, the output probability for each instrument should be independent from the other instruments. To achieve this, instead of the softmax, the *sigmoid* function is used. This function allow us to have each value node into the $[0, 1]$ range, but the sum of all the probabilities may not equal to 1. This is the intended behaviour, as each probability does not influence the other probabilities. The sigmoid activation function is described as $g : \mathbb{R}^{N_Z} \to \mathbb{R}^{N_Z}$:

$$g(\mathbf{z})_j = \frac{1}{1 + e^{-z_j}} = \frac{e^{z_j}}{e^{z_j} + 1} \qquad \text{for } j = 1, \ldots, N_Z, \tag{4.4}$$

where $\mathbf{z}$ is the input vector and N_Z is the dimensionality of $\mathbf{z}$.

The output of the described pipeline is a set of arrays $X_p = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_{N_S}\}$. Such set represents the probability for a clip to belong to each instrument class $m_i \in M$ with a time-varying prediction for each frame s_i. In fact, for each $\mathbf{x}_s \in X_p$, we have a value $x_{s,m}$ for each musical instrument m. However, our output domain consists in a single prediction for each instrument m_i. In order to effectively pool these prediction and output a single prediction for each instrument m_i, we use the adaptive softmax pooling layer developed by McFee et al. [51].

Given the set X_p, the *ground truth* label value h_m for a single instrument on a given excerpt, a likelihood function for the entire clip $\hat{L}(h|X_p)$, a likelihood function for the single frame $\hat{l}(h|x_s)$, the probability value $\hat{p}_m$ for a single instrument m can be computed through $\hat{L}$, as:

$$\hat{L}_{\alpha_m}(h_m|X_p) = \sum_{\mathbf{x}_s \in X_p} \hat{l}(h_m|x_{s,m}) \left(\frac{\exp(\alpha_m \cdot \hat{l}(h_m|x_{s,m}))}{\sum_{\mathbf{q}_s \in X_p} \exp(\alpha_m \cdot \hat{l}(h_m|q_{s,m}))} \right), \quad (4.5)$$

where $\hat{L}_{\alpha_m}(h_m|X_p) = \hat{p}_m$ is the predicted value for the m-th instrument. The final output of the model will be a vector of *class probabilities* $\hat{\mathbf{p}}$. Each weight $\alpha_m \in \boldsymbol{\alpha}$ is a free parameter for the m-th instrument and it is learned by the model alongside its general parameters $\mathbf{w}$. As it is continuously differentiable, the adaptive pooling updates the weights for each parallel sub-network of the classifier, following the formula provided in the section 3.3.5 for the *adaptive pooling*. Finally, as we want to prevent the overfitting of the system, we employ the regularization technique for this layer. The *regularized auto-pool* (RAP) approach applies a penalty to $\boldsymbol{\alpha}$ to prevent the model from applying too much weight on individual instances, but without an explicit bound to the maximum weight. The objective function of the model can now be formulated as:

$$\min_{\mathbf{w},\boldsymbol{\alpha}} J(\mathbf{w}) + \lambda ||\boldsymbol{\alpha}||^2. \quad (4.6)$$

We use the quadratic penalty $||\boldsymbol{\alpha}||^2$, so that the penalty quickly grows with $\boldsymbol{\alpha}$. This effectively promotes a mean-like behaviour, but still provides flexibility to learn a max-pooling behaviour if necessary.

Having pooled the results of the parallel feedforward network, the final output of the model is simply an array of N_M probabilities, one for each instrument. Now we can threshold the *class probabilities* and compute the loss function, or, if we finished the training procedure, make predictions on new, unseen samples.

Chapter 5

Experimental Results

In this section we discuss the implementation of the models and we review the experimental results. Firstly, in section 5.1 we discuss the metrics used to assess our method. Then, in section 5.2 we review the results on the OpenMIC-2018 dataset, describing the different approaches we employed and the survey we set-up to enlarge the OpenMIC-2018 dataset. In section 5.3 we compare the state-of-the-art and out method on the IRMAS dataset. Finally, in section 5.4 we show our results on the MedleyDB + Mixing Secrets dataset and we compare the results obtained with the ones from the state-of-the-art.

5.1 Evaluation metrics

In order to validate of the model, we cannot only refer to the loss function $J(\mathbf{w})$ computed by the network. In fact, as we are in a classification problem, the loss function is usually not easily human interpretable in order to understand effectiveness of the model on new samples. We need other metrics to assess the classification on the output space. In the literature there are popular metrics, which are usually computed class-wise and then averaged, to have a perception of the overall classification results. *Accuracy* is one of the standard metrics, defined as:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{N_p}, \tag{5.1}$$

where TP is the number of True Positives and TN is the number of True Negatives, and N_p is the total number of samples. *True Positives* are samples classified by our model and labelled in our *ground truth* as *Positives*. A positive sample for the instrument m_i is a sample where the instrument m_i

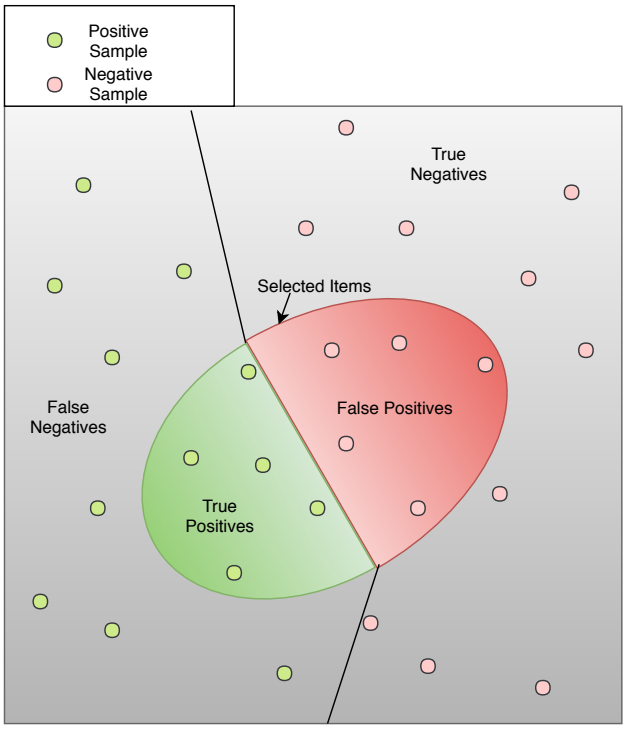


Figure 5.1: Graphical representation of a generic classification task.

is *present*. *True Negatives* are samples classified and labelled as *Negatives*. A negative sample for the instrument m_i is a sample where the instrument m_i is *absent*. In figure 5.1 we show a graphical representation of a generic classification task with examples for *True Positives*, *False Positives*, *True Negatives*, *False Negatives*. *False Positives* are samples classified as *Positives* but labelled as *Negatives*, and *False Negatives* are samples classified as *Negatives* but labelled as *Positives*. The items in the coloured ellipse are classified as positives. Therefore, based on their true value, they are either false positives or true positives. The same applies for items outside of the ellipse, but with their negative counterpart. These are the samples classified as negatives.

The accuracy, however, can be misleading in case the dataset is strongly biased towards one class. In this case, the *Accuracy* might provide an outstanding result even if the classifier learns to classify every sample as the majority class. To tackle this problem, other metrics are often employed, such as *Precision* and *Recall*, formulated as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{5.2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{5.3}$$

where TP is the number of True Positives, FP is the number of False Positives, and FN is the number of True Negatives. The *precision* can be seen as how good the model behaves for predicting true positive samples while not predicting many false positives. If the model predicts 0 false positives, the precision is at 1. The *recall* can be seen as how good the model behaves for predicting true positive samples while not predicting many false negatives. If the model predicts 0 false negatives, the recall is at 1.

In order to combine the information of those two metrics into one, we use the F_1 score. It is defined as the harmonic average of the *Precision* and *Recall*, where an F_1 score reaches its best value at 1 (perfect precision and recall) and worst at 0. It is defined as

$$F_1 = \left(\frac{\text{Recall}^{-1} + \text{Precision}^{-1}}{\beta} \right)^{-1} = \beta \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{5.4}$$

with $\beta = 2$. In our model, each musical instrument m_i is a class. As we can compute these metrics for each class, an aggregation technique is requested in order to crunch the values. We have two possibilities. We can use the *macro* average of our results, which is the average of the metrics over the classes. Alternatively, we can use the *micro* average of our results, which is a weighted average of the metrics over the classes with the weights directly proportional to the number of samples of the class. This metric is more useful in the case a few classes makes up the majority of the dataset, and it is more important to have a good performance over them rather than on the classes with a lower number of samples.

5.2 Experiments on OpenMIC-2018 dataset

Here we review all the experiments conducted on the OpenMIC-2018 dataset [38]. The dataset presents problems regarding the completeness of the labels. In fact, for each audio excerpt we do no not have a label for each one of the 20 musical instruments, as we will explain in section 5.2.1. Due to this problem, we applied different approaches for our classification. First of all, as the baseline classification for the dataset is computed on the binary classification for each class, which for each instrument aim at classify only its presence or absence without focusing on other instruments, we replicated these settings (section 5.2.2). Then, in section 5.2.3, we use all the labels of the dataset together in a multi-class multi-label classification problem.

After that, we set-up a survey in order to obtain labels for each musical instrument on each clip, and we aggregated the results using 2 different approaches, which we will explain in section 5.2.4. This was necessary in order to perform the classification task in a *multi-label* setting where we could exploit the information of the labels for each instrument on each clip. We then evaluated the multi-class multi-label classifiers on the data collected through the survey. These results are described in section 5.2.5 and section 5.2.6. All the approaches were implemented and evaluated using Python[1] programming language. For audio processing and manipulation we employ the Librosa [50] library. We used the Keras[2] framework alongside TensorFlow [1] libraries for the development and implementation of Deep Learning networks.

5.2.1 Dataset composition

The OpenMIC-2018 is a new, open dataset recently released by Humphrey et al. [38]. The dataset was published in order to satisfy the requirements for a large, diverse, polyphonic and multi-label open access dataset, for the specific task of *Musical Instrument Recognition*. The dataset contains 20000 audio clips of 10 seconds of duration. The audio clips are annotated for 20 musical instruments: accordion, banjo, bass, cello, clarinet, cymbals, drums, flute, guitar, mallet percussion, mandolin, organ, piano, saxophone, synthesizer, trombone, trumpet, ukulele, violin and voice. However, as one could notice from figure 5.2, the total number of clips (20000) does not match the number of labels per musical instrument. In fact, due to the difficulty of annotating each clip for all the musical instruments, as it is a very costly process for human workers, Humphrey et al. decided to release the dataset with only 3 or 4 labels per clip. The musical instrument labels change from clip to clip. This has two consequences:

- Given a clip, we do not have any information about the presence or absence of instrument a part of the 3 or 4 that are annotated.
- The 3 or 4 instruments annotated changes from clip to clip.

Moreover, the clips are labelled in a *weakly* fashion, which implies that a musical instrument is annotated as present if and only if it is active in almost a part of the 10 seconds clip.

[1] `https://www.python.org/`
[2] `https://keras.io/`

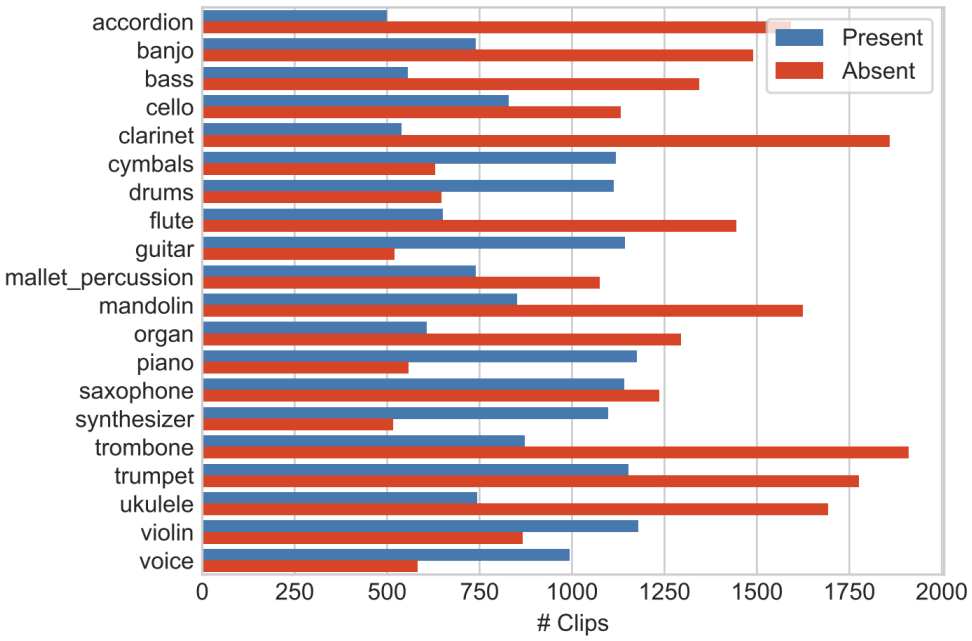


Figure 5.2: Distribution of annotations for each musical instruments in the OpenMIC-2018, taken from "OpenMIC-2018: an open dataset for multiple instrument recognition" by Humphrey et al. [38].

5.2.2 Results on Binary Classification

As the dataset is not suitable for a direct multi-class multi-label problem formulation, due to the lack of some musical instrument labels in each clip, we conducted the first experiments for binary classification. In this section we will explain our procedure. We trained a network for each musical instrument, and compared the results of F_1 scores with the baseline classification.

Moreover, for the purpose of having another benchmark of reference on this dataset, we compare our model with another architecture we refer to as *attention model*. This model is similar to our proposed in chapter 4. It is a feedforward network which employs the VGGish features as input, but comprises more dense block layers such as the ones described in section 4.1.2 (between 3 and 5, depending on the particular instance). After the dense blocks we use the *attention pooling* technique described in section 3.3.5, in a *multi-level attention* fashion such as the one proposed by Yu et al. [61].

Let us describe the hyperparameters we used for the training procedure of our model. We employed the *binary cross-entropy* as loss function, which is the most common choice when performing binary classification, and trained the model on *batch sizes* of 128 samples. As the model is for a sin-

Instrument	#Train	#Test	Instrument	#Train	#Test
accordion	1533	538	mandolin	1837	627
banjo	1740	478	organ	1459	431
bass	1425	463	piano	1305	415
cello	1464	485	saxophone	1736	629
clarinet	1745	640	synthesizer	1222	380
cymbals	1299	436	trombone	2040	720
drums	1323	424	trumpet	2131	785
flute	1522	562	ukulele	1835	590
guitar	1214	436	violin	1402	631
mallet percussion	1324	478	voice	1190	374

Table 5.1: Number of training and test set samples for every musical instrument, employed for the binary classification task.

gle musical instrument, and the clips are 10 seconds long, the input matrix extracted from the VGGish model will be $128 \times N_S$ with $N_S = 10$, and the output will be a single probability value $\hat{p}$ for the clip, as $N_M = 1$.

The output $\hat{p}$ is the probability value for the sample to belong to the class learned from the model. The *class* coincides with the instrument m_i we are classifying in the model. Each set labelled for the musical instrument has been divided into a *training* set and a *test* set, and we report the partitions for each musical instrument in table 5.1.

For each musical instrument we used a *validation* set, comprised of the 10% of the *training* set. The validation set is of extremely importance as we employed it for tuning *hyperparameters* during the optimization process, such as the learning rate and the number of epochs. We used a *stochastic gradient descent* optimizer [7] with a *learning rate* of 0.01. In order to avoid *overfitting*, when the validation loss stops to decrease during the training procedure, the *learning rate* is reduced of a factor of 0.1. The model is trained for 100 *epochs*, which are a full iteration of the dataset, but the *early stopping* technique is employed. When the validation accuracy stops to increase, the training procedure is halted. Let us remember the model architecture from chapter 4. In this case, the first dense layer and the second dense layer both have 4096 hidden units. For both of them, we apply a *dropout* of 50%. For the third layer, a *sigmoid* activation function is employed, as the classification is on a single musical instrument. Then, the *adaptive pooling* or *autopool* layer employs a regularization technique with a regularization rate $\lambda = 0.0001$.

As aforementioned, the model will output a single value $\hat{p}$ which is the *class probability*. The *class probability* is the probability, assigned by the

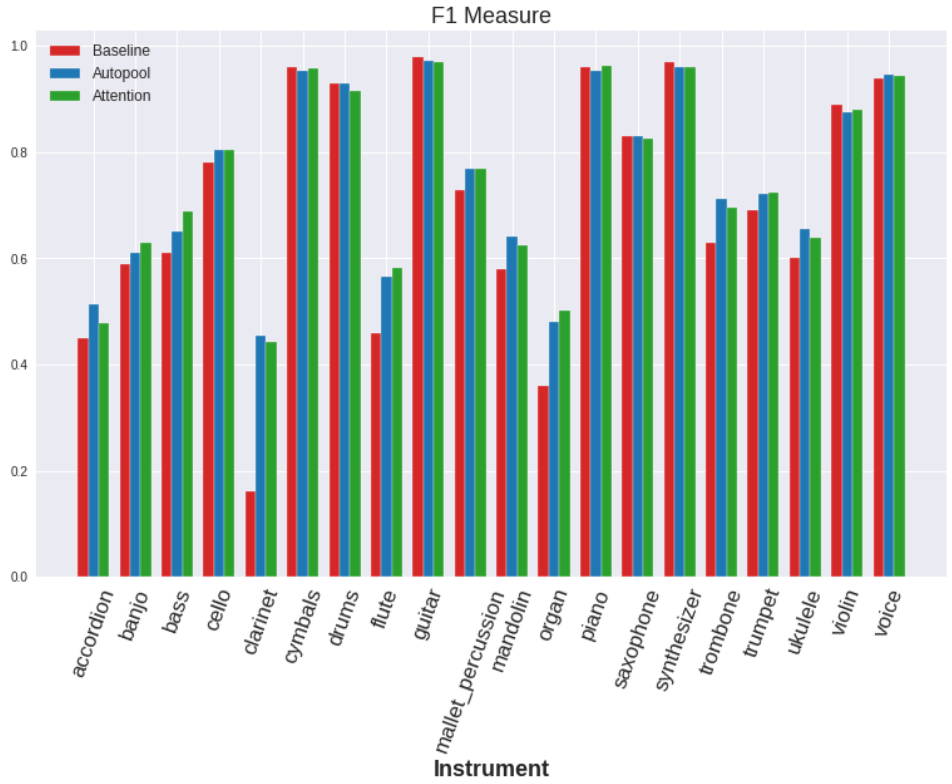


Figure 5.3: Comparison of the binary classifiers.

model, of a musical excerpt to belong to a class m_i. In order to evaluate the results, we will threshold the value with a threshold $\theta = 0.5$, in order to get binary levels of 0 and 1. Now, we can compare the thresholded predictions against the *ground truth* labels and compute the metrics described in section 5.1.

The results for each musical instrument are presented in figure 5.3. We show the F_1 score as it is a combination of *precision* and *recall*. Our model greatly outperforms the baseline of the dataset for *clarinet* and performs better in other classes as well, such as *flute*, *organ*, *mandolin*, *ukulele* and *trombone*. For musical instruments where the baseline model is already effective, for example *voice*, *guitar* and *piano*, we found no improvements. This is probably due to the abundant number of positive examples we have for these classes, as we can see from figure 5.2, which bring the baseline model to effectively learn the representation of the instruments. The numerical version of the results for each class of our model are listed in Appendix A. As aforementioned, we compare our *autopool* model with another model based on the *attention pooling* method. The results are close to each other. The

proposed and the attention model both achieved an average F_1 score on the classes of 76%, against the baseline average of 70%. However, in the case of the *attention* model, we fine-tuned the parameters to build the architecture for each specific musical instrument, whereas with the *autopool* model we used the same architecture in each instance. In fact, for each *attention* model we researched the exact parameters for building the architecture in order to optimize the results on the specific instrument. This brings us to the conclusion that the *autopool* model posses better generalization capabilities than the other architecture.

5.2.3 Results with masked input

It is of our interest to build a single model which is able to identify all of the 20 musical instruments of the described dataset. However, as we described before, a direct formulation of the multi-label problem is not possible, due to the absence of the labels for the majority of the musical instruments in each musical excerpt. However, we found a solution which allows us to formulate a multi-label problem exploiting the uncertainty of the unknown labels. For this purpose, we created a *masked loss function*, which is a variation of the loss function where the *backpropagation* of the error can be blocked for some inputs. For example, let us imagine we feed a sample where we have a label for the class m_i and the class m_j, but we have no label for the class m_k. In this configuration, we would like to back-propagate the error only for class m_i and m_j. For the class m_k, we do not compute the error as it is useless, and we do not consequently update the weights of the network. The model will however make a prediction, but we will not compare it with the output, so it is "invisible" to the update of the model. Following this assumption, we built a *masked binary cross-entropy* loss function and consequently a masked version of the metrics. To indicate the absence of the label to the model, we modified the *ground truth* labels by using 0 for a negative label, 1 for a positive label, and -1 for a missing label. The masked function recognizes the missing label and blocks the error backpropagation in that case. Summarizing, we will use all the audio clips of the dataset as input, and the outputted *class probability* for each instrument will be taken in consideration for the update of the *weights* if and only if we have a label for that instrument.

Regarding the hyperparameters used for the training procedure, we employed the *masked binary cross-entropy* as loss function, and trained the model on batch sizes of 128 samples. We employed the Stochastic Gradient Descent optimizer with a starting *learning rate* of 0.1. The learning rate has

an automatic decay of factor 0.001 for each epoch. We employed a training set of 14915 samples and test set 5085 samples, which are the same clips described in table 5.1, but all combined together. The 10% of the training set was used as validation set. The training procedure is launched for 60 epochs, but we employ the validation loss to avoid overfitting. If the validation loss stops to decrease, the training procedure is prematurely halted.

As in the previous case, the input matrix will be of dimensions $128 \times N_S$, with $N_S = 10$. The architecture has 4096 hidden units for both the first two layer, with a *dropout* of 30%. The third dense layer employs a sigmoid function for the activation of the neurons, because every musical instrument is equally probable to be present, as described in section 4.1.2. Then, the *autopool* layer is employed, with a regularization rate $\lambda = 0.0001$. The output is a vector of 20 elements, $N_M = 20$. Hence, the model will output a vector $\hat{\mathbf{p}}$ for the *class probability*. To evaluate the results, we will threshold the value with a threshold $\theta = 0.5$, and we obtain binary levels of 0 and 1. Now, we can compare the predictions against the *ground truth* labels, when the labels are different from -1, and compute the metrics described in section 5.1.

Results in figure 5.4 show similar performances of the binary classification described before. The model works particularly better than the baseline model on *clarinet*, *organ*, *flute*. No improvements for musical instruments like *guitar*, *drums* and *voice*, with respect to the baseline model. This is due to the high number of positive examples for these classes, as mentioned before. The proposed model achieved an average F_1 score of 76% for the *macro* average, against a 70% average of the baseline model. We show the numerical results for the proposed method and the baseline for each musical instrument in Appendix A.

In order to better analyse the performances of our model, we introduce a *modified confusion matrix* developed by Gururani et al. [31]. The classic confusion matrix shows the frequency of confusion between every pair of predicted class label and the *ground truth* class label. However, this formulation is possible only if we are in a multi-class single-label context, where each sample belongs to one and only class. In our case, every possible combination of the N_M labels is possible, therefore the classic confusion matrix visualization is not effective. However, as a confusion matrix is an intuitive approach to gain insight of the model, we use a *modified confusion matrix*. When checking the misclassification of the model, we are interested in inspecting if a musical instrument is often mistaken for another. In order to find this kind of error, we must check when the model does not classify a generic musical instrument m_i which is labelled as present (false negative),

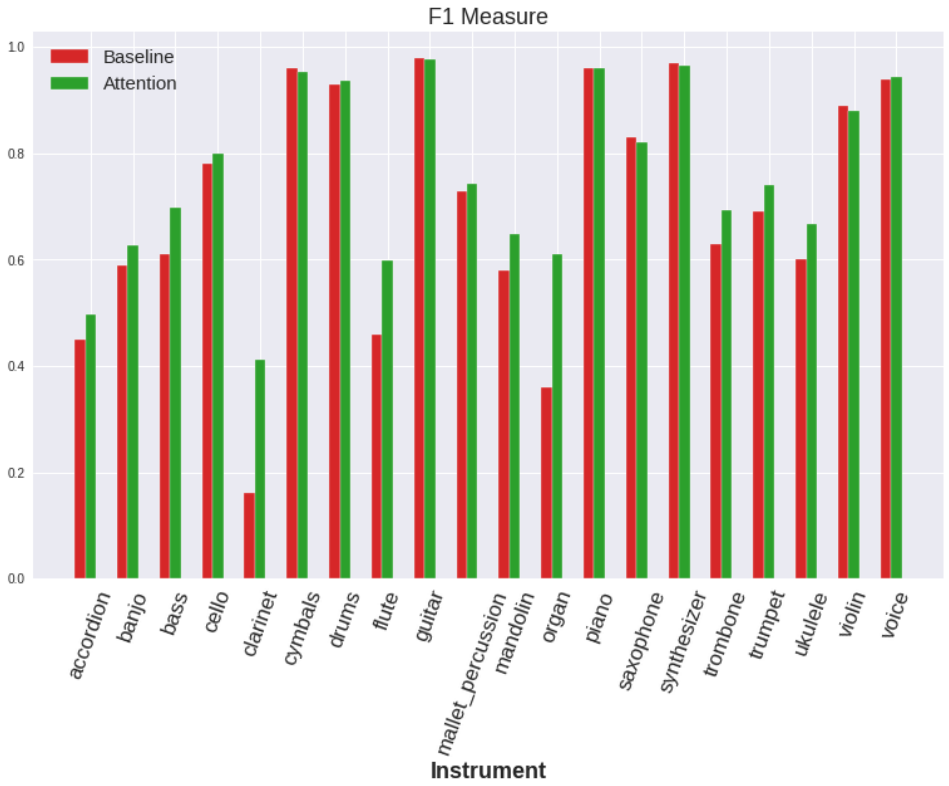


Figure 5.4: Baseline comparison of the classifiers.

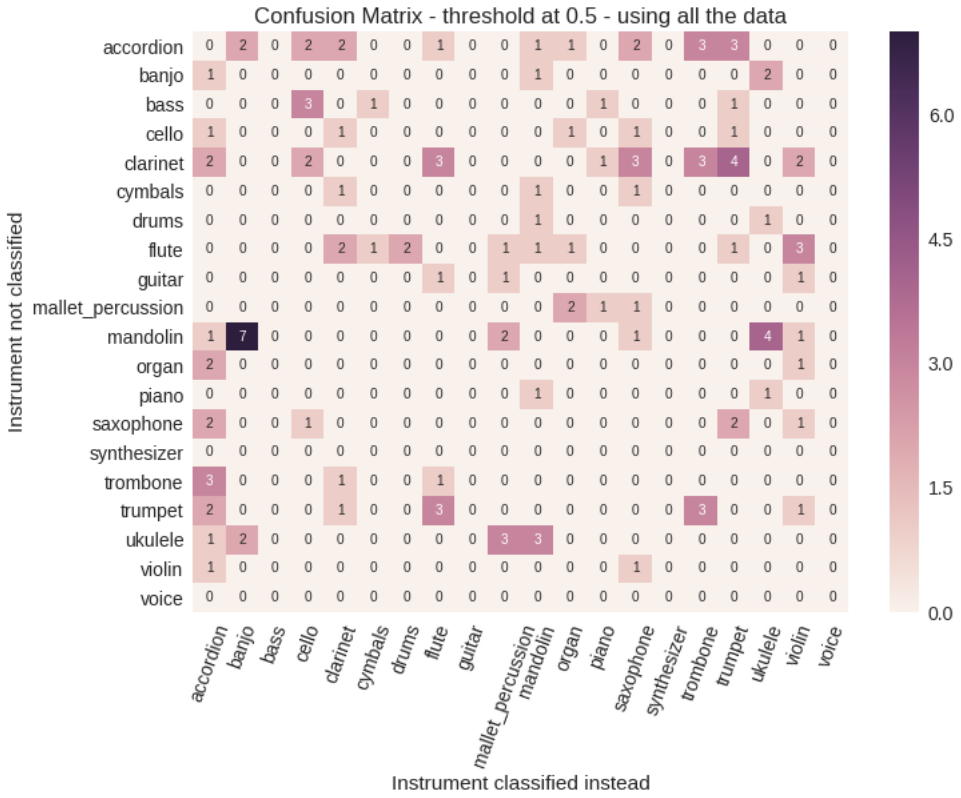


Figure 5.5: Modified confusion matrix on the complete dataset.

but classify a generic musical instrument m_j instead which is labelled as absent (false positive). So, in our $N_M \times N_M$ matrix we show how many time a false negative for the musical instrument m_i occurred at the same time of a false positive for the musical instrument m_j. We show the result of this comparison in our modified confusion matrix in 5.5.

In this case, it is evident how the model has difficulties in distinguishing *ukulele*, *mandolin*, *banjo*, which are all string instruments with similar timbres. For the same reason, the model struggles to distinguish wind instruments such as *clarinet*, often confused for *trombone*, *trumpet* and *saxophone*. However, the model performs really good with instruments like *voice*, *synthesizer*, and is able to effectively distinguish the *violin* and the *cello*, which are difficult to identify even for a human listener.

5.2.4 Survey

The OpenMIC-2018 [38] dataset is annotated for 20 classes of musical instruments. However, due to the high cost of annotating clips for 20 musical

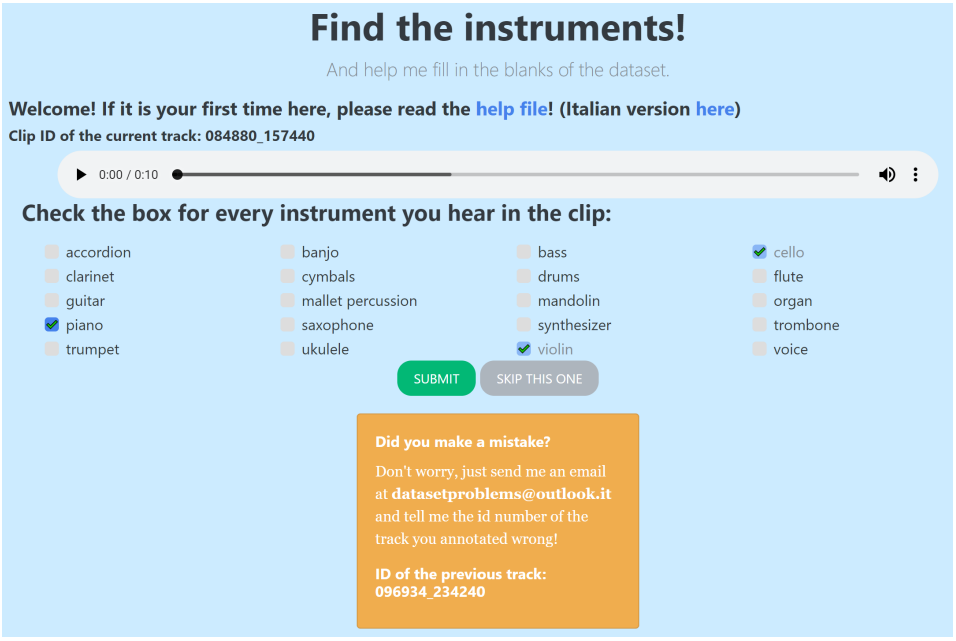


Figure 5.6: Web page for the survey procedure. In this example, the clip was already annotated for the presence of violin and cello, and the user added the piano label.

instruments, even in a *weakly* fashion, each audio clip has been annotated only for 3 or 4 musical instruments out of 20. The annotated musical instruments changes from clip to clip. This implies that the subset of annotated labels for each clip changes within the dataset. This can be a problem, as in order to compute *multi-label* classification, each clip must have a ground truth label for each class. Otherwise, we cannot confront the output value of a multi-classifier with each outputted probability value and correctly back-propagate all the errors. In order to solve this issue, we decided to annotate a subset of the audio clips through *crowd-sourcing*, by the means of the web-survey. We took a subset of 1000 random clips, and we ensured that at least 50 examples where the instrument m_i is present (which we define as a *positive* example) were included. We asked 68 participants to listen to random audio clips of the subset and to label them with the presence of the active instruments. In figure 5.6 we show the front-end of the developed web survey. We implemented the web survey using HTML, CSS, JavaScript and PHP programming language. We can see that there is an audio player, from which the user can listen to the audio clip, and a check-box for each musical instrument. The task for the user is to tick the check-boxes for the musical instruments he can recognize as present in the audio clip. When the instrument is active for at least a recognizable fragment of the musical excerpt, it

is considered present. We exploited the information already present in the dataset, in order to ease the task for the users. In fact, when a musical instrument was already labelled, we showed it to the user through an already ticked box, which he cannot un-tick, as we can see for the *violin* and the *cello* class in figure 5.6. When the user is satisfied with the annotation, he can submit the data. For all the check-boxes the user did not tick, the clip is considered as a *negative* example for the corresponding musical instrument, which means the instrument is absent. If he/she finds the clip too difficult to label, he/she can skip the clip and he/she will be assigned a new random one. At the bottom of the web-page there is a 10 second clip example for every musical instrument, in order to help the users to remember the timbre of the musical instruments. Each session of annotations does not have a fixed duration. The user can decide how many clips to label.

We collected 2000 annotations, which implies there are 2 annotations for each audio clip. In order to enforce the consistency of the survey, every audio clip was never annotated twice by the same user. However, since we only have 2 values for each clip, we cannot employ majority voting to produce the final results of the survey, as the number of votes will often be even. To overcome this problem, we decided to employ two different strategies for aggregating the results. One is to generate a *union* subset of the labels, which means that all the labels annotated by each user are considered. The second one is to generate a *intersection* from the results, such that a label on a clip must be annotated for each user in order to be recognized as valid.

Generally speaking, the *union* approach may generate false positives as some user could erroneously tag absent musical instruments that are present. However, the approach is robust towards false negatives. Instead, the *intersection* approach may produce false negatives, but is robust versus false positives, as multiple users must label the same musical instrument, in order to confirm the label as present.

In figure 5.7 we can observe how the two different aggregating strategies affect the results of the survey and the number of samples for each of the class. As expected, the *intersection* strategy is more selective and produces less positive examples than the *union* strategy. Some musical instruments are consistent in both the solutions, as they probably appear less often in the clips, such as the *accordion* or the *ukulele*. For other musical instruments, like the *bass*, the number of samples almost doubles in the union scenario. This is probably due to the listening inexperience of users, which easily missed the musical instrument activity in many clips. The *synthesizer* instrument also has many more classified samples in the union case. This is probably due to the instrument having a fuzzy definition, that users

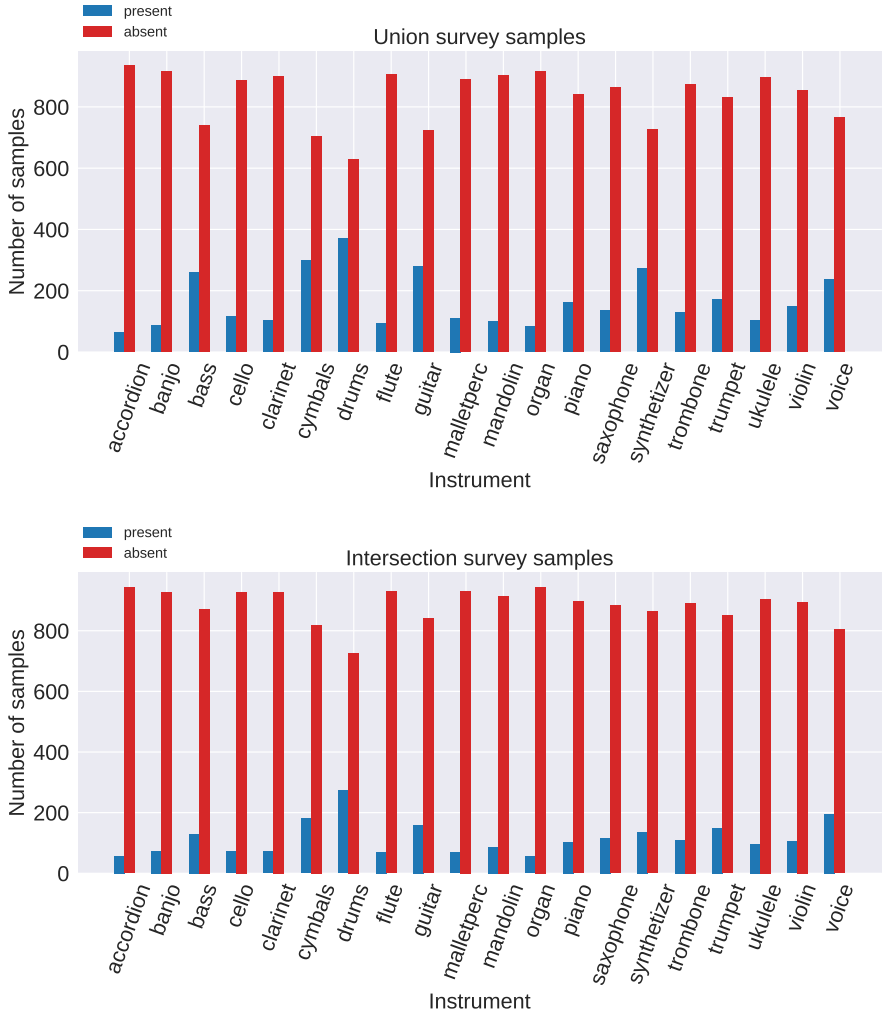


Figure 5.7: Results of the survey, with aggregated results with respect the two strategies, union and intersection.

interpreted deliberately.

5.2.5 Results on the Union of the Survey

After collecting the results of the survey, as described in the previous section, we employed two different aggregation strategies and performed the training procedure of the model on the data. Here we introduce the results using the *union* aggregation strategy. In this case we have a complete set of labels for each clip, allowing us to formulate a multi-class multi-label problem in the proper way. Due to the low number of clips however, the network could have difficulties in learning the correct mapping. Thanks to the representation provided by the VGGish architecture, our proposed network manages to effectively complete the training procedure on the small dataset.

We used a subset of 800 songs for the training procedure, and we used the remaining 200 songs for the testing procedure. The proportion of the positive musical instruments' samples in the test set is kept the same in the testing set. Due to the low number of samples, we did not employ a validation set, and we scheduled the training for 80 epochs. We used the *binary cross-entropy* as loss function, and a Stochastic Gradient Descent optimizer with an initial *learning rate* of 0.1, reduced after each epoch of a factor 0.001. We employed a batch size of 32 samples. The architecture has 4096 hidden units for the first and the second layer, with a *drop rate* of 30%. The sigmoid activation function is applied to the third dense layer, and after that the *autopooling* layer pools the predictions and outputs a vector of length $N_M = 20$. We applied *regularization* to the pooling layer with a regularization rate $\lambda = 0.0001$. The model will output a vector $\hat{\mathbf{p}}$ for the classes probability. In order to evaluate the results, we threshold the values with a threshold $\theta = 0.5$, to generate a binary level.

The results are shown in figure 5.8. We compared the F_1 scores of our proposed model with a model using the multi-attention module as the one described in the previous section 5.2.2. We show the numerical results for each musical instrument in Appendix A. The proposed model performs slightly better in the majority of the musical instruments, and definitely outperforms the attention model for the *clarinet* instrument. The proposed model performs better than the attention counterpart, achieving a macro average F_1 score of 79% and a micro average F_1 score of 82%, while the attention model achieved, respectively, a score of 77% and 80%.

In figure 5.9 we show the *modified confusion matrix* obtained from the testing procedure. We can see that the model mistakes string instruments. For instance, it frequently maps the *guitar* instrument as a *bass*, due to

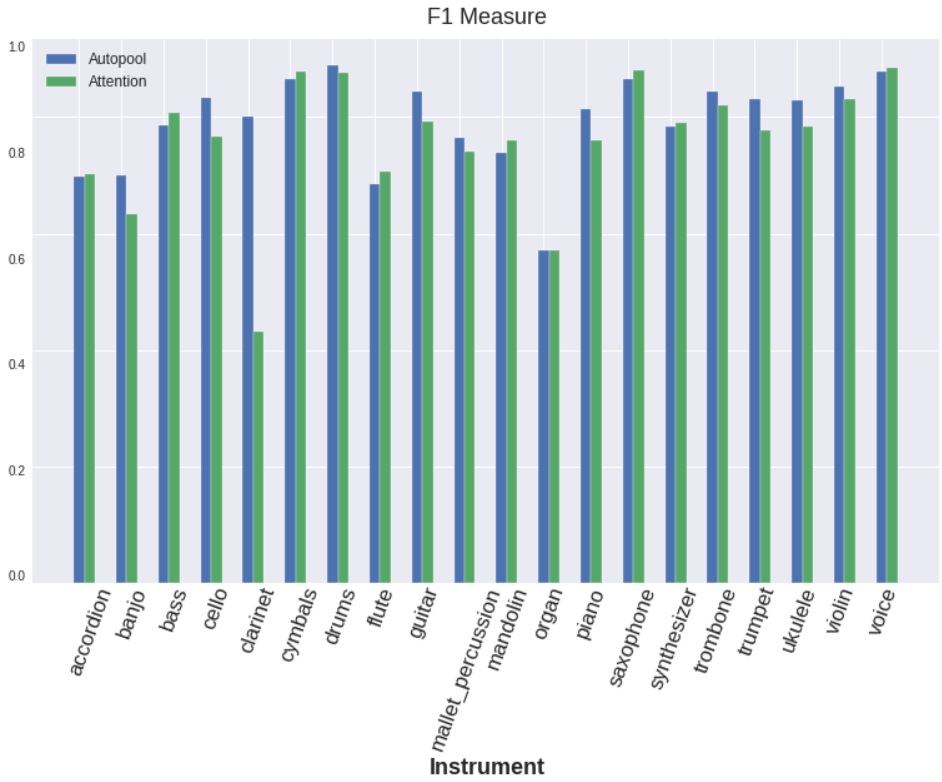


Figure 5.8: Comparisons of the results on the union strategy of the survey data.

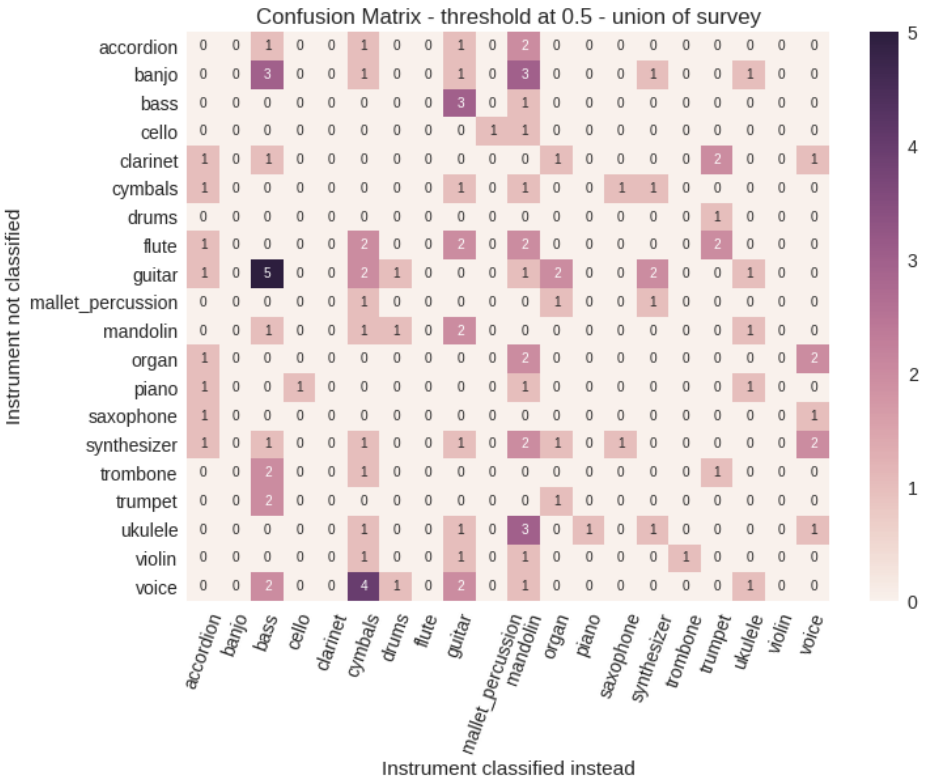


Figure 5.9: Modified Confusion Matrix on the union of the survey data.

the similarity between the timbres. For the same reason, *banjo* and *ukulele* are often mistaken for a *mandolin*. *Voice* has been mistaken for *cymbals* 4 times, even if their timbres should not be similar. This may be due to the low number of samples used for training and testing. The model effectively distinguish similar classes such as *violin* and *cello*, or *trumpet*, *trombone* and *saxophone*, which are difficult to separate even for a human listener.

5.2.6 Results on the Intersection of the Survey

Here we discuss the results for the training procedure performed on the results of the *intersection* strategy. In this case we also have a complete set of labels for each clip, allowing us to formulate a multi-class multi-label problem correctly.

We performed the training procedure using a subset of 800 songs for the training set, and the remaining 200 songs for the testing set. These are the same of the union case. We did not employ a validation set. We employed the Adam [41] optimizer, as it resulted in better performances with respect to the Stochastic Gradient Descent optimizer on this dataset, and trained the model on batches of 128 samples, which had its initial *learning rate* of 0.01 reduced of a factor 0.1 every time the loss function stopped to decrease. We used the *binary cross-entropy* as loss function for the optimization. The training procedure was scheduled for 80 epochs. The architecture has 4096 hidden units for the first dense layer and 2048 units for the second dense layer, with a *drop rate* of 50%. The rest of the architecture is the same as the *union* case: a sigmoid activation function is employed for the third dense layer, after which the *autopooling* with a *regularization* rate $\lambda = 0.0001$. The *autopool* layer pools the predictions and outputs a vector of length $N_M = 20$. As before, the model will output a vector $\hat{\mathbf{p}}$ of *classes probability*. To evaluate the results, we threshold the values with a threshold $\theta = 0.5$, in order to get binary level. We can then can compare the thresholded predictions against the *ground truth* labels and compute the metrics.

In figure 5.10 we compare the F_1 scores of our proposed model with a model using the multi-attention module as the one described in the previous sections. We show the numerical results for each musical instrument in Appendix A. The proposed model outperforms the attention model in most of the musical instruments. However, the attention model seems to be scoring better results for brass instruments such as *trumpet*, *trombone* and *saxophone*. The proposed model achieved a macro average F_1 score of 80% and a micro average of 81%, while the attention model achieved, respectively, a score of 76% and 78%.

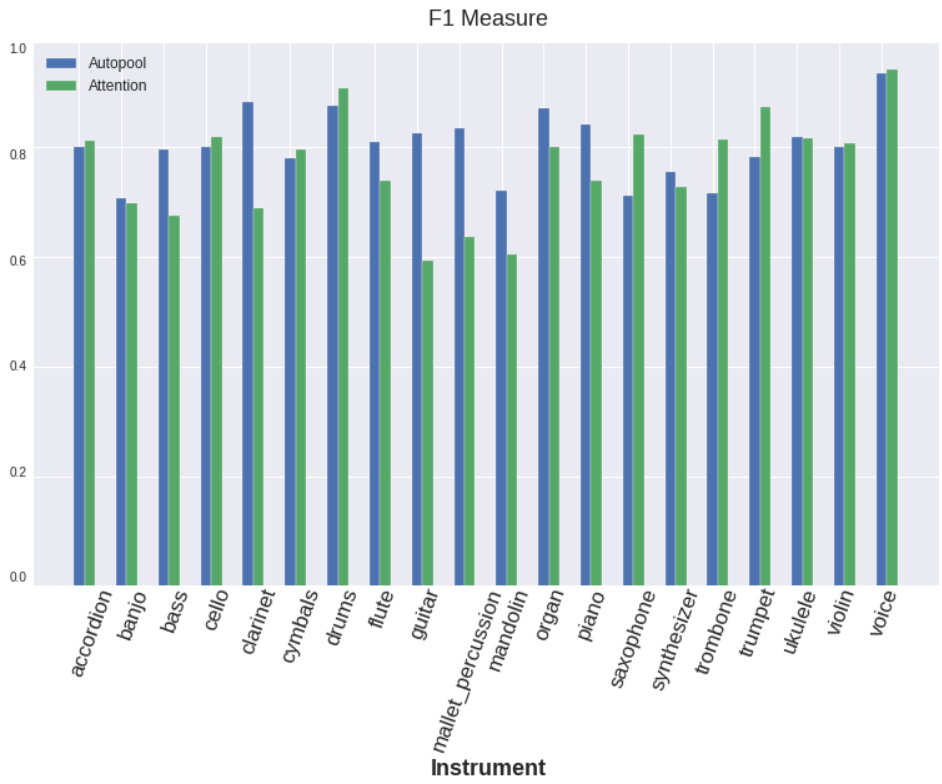


Figure 5.10: Comparisons of the results on the intersection of survey data.

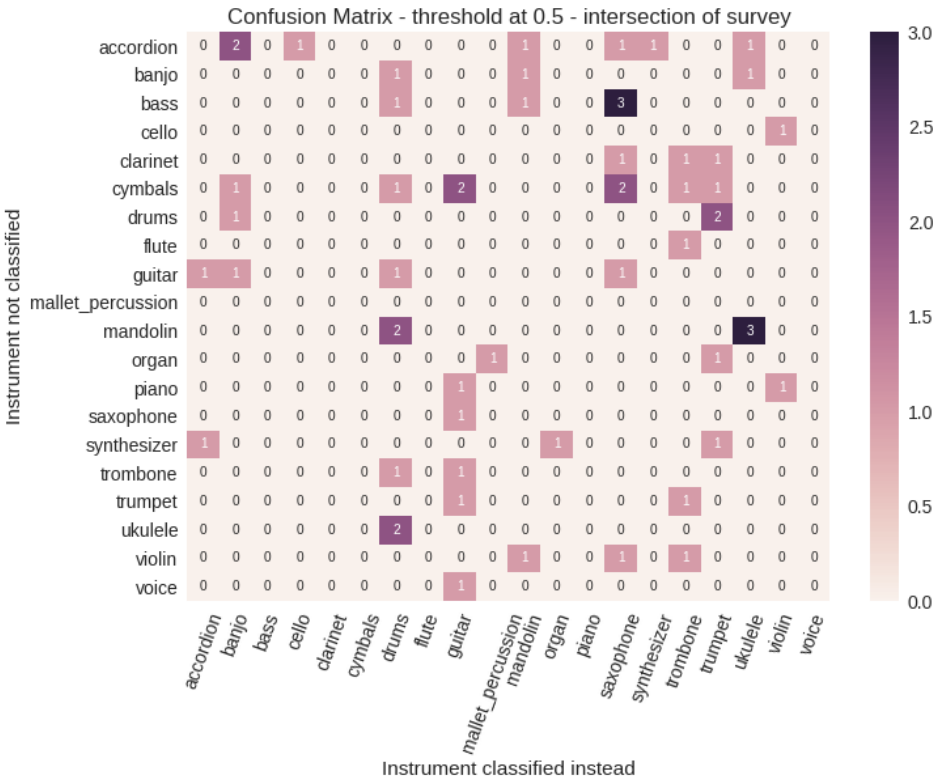


Figure 5.11: Modified Confusion Matrix on the intersection of survey data.

In figure 5.11 we show the *modified confusion matrix* obtained from the evaluation of the model. The *mandolin* class is often mistaken for the *ukulele* class due to the similarity between the timbres. The model shows mistakes with respect to the *bass* class and the *saxophone* class. This particular mistake could originate from the lack of enough samples to properly train the model. The model performs very good on the *bass*, *piano*, and *voice* class. It is able to effectively distinguish the *violin* and the *cello* class.

5.3 Results on the IRMAS dataset

For the best of our knowledge, no studies have been published in the literature on the OpenMIC-2018 dataset. For this reason, we decided to test our model on other datasets in order to evaluate its effectiveness. We compare it against other state-of-the-art works. On the IRMAS dataset [6], the current best performing works are the CNN model proposed by Han et al. [32] an the CNN models proposed by Pons et al. [53]. These works focus on a training set which is annotated with a *predominant* instrument, but they perform the testing phase on a set annotated for multiple instruments, using an aggregation strategy we will describe later in this section.

The data available from the dataset are already divided in a training set and a testing set. We use the same dataset splits employed by the other works in the literature. The training set consists in 6705 audio excerpts of 3 seconds, labelled with a single *predominant instrument*, which is the most easily recognizable instrument playing in the excerpt. The testing set consists of 2874 audio excerpts of length varying between 5 and 20 seconds. For these clips, each musical instrument is labelled. There are 11 labelled pitched musical instruments: cello (cel), piano (pia), violin (vio), clarinet (cla), organ (org), acoustic guitar (gac), trumpet (tru), flute (flu), saxophone (sax), electric guitar (gel) and voice (voi).

We formulated the problem in a classic *multi-class multi-label* fashion, but since the training set is not *multi-label* we had to take some precautions. In the training procedure, we employed the *categorical cross-entropy* and we used the *softmax* activation function in the prediction layer of the network, due to the training set being labelled for only one musical instrument. The *categorical cross-entropy* is used in multi-class single-label settings, and the softmax function attributes the highest probability only to the most probable class. We employed a Stochastic Gradient Descent optimizer with an initial *learning rate* of 0.0001. The 10% of the training set was used as a validation set. The training procedure was scheduled for 300 epochs, but *early stopping* was employed to halt the training phase when the validation

	Micro			Macro		
Model	Precision	Recall	F1	Precision	Recall	F1
Han et al.	0.655	0.557	0.602	0.541	0.508	0.503
Pons et al. - Single	0.611	0.516	0.559	0.523	0.480	0.484
Pons et al. - Multi	0.650	0.538	0.589	0.550	0.525	0.516
Proposed	0.657	0.592	0.623	0.569	0.575	0.541

Table 5.2: Comparison of results on IRMAS Dataset.

accuracy stopped to increase. When the validation loss stopped to decrease, the learning rate was reduced of a factor 0.1. Regarding the architecture, the first two dense layers both have 2048 hidden units, with dropout rate of 50%. The *autopool* layer has a regularization rate $\lambda = 0.1$. The input matrix is $128 \times N_S$ with $N_S = 3$ and the output is a vector of *class probabilities* $\hat{\mathbf{p}}$ of $N_M = 11$ elements.

For the testing phase, we divided the test segments into chunks of 3 seconds, as the input of our model is constrained to this length, and obtained the predictions for each 3 seconds segment. Then, for properly evaluating the results, we used the aggregating procedure described by Han et al. [32]. We summed all the outputs $\hat{\mathbf{p}}$ class-wise over the whole audio clip (which is more than 3 seconds) and normalized the values by dividing them with the maximum value among the classes, such that the values were scaled between zero and one, then we thresholded them. Then, we computed the metrics. The threshold yielding the best results was of value $\theta = 0.45$.

Results are summarized in table 5.2. We achieved better F_1, *precision* and *recall* scores than the other works. The proposed model achieved an average F_1 score of 62% for the *micro* average and of 54% for the *macro* average. We list the results for each musical instrument in Appendix B. In figure 5.12 we show the *modified confusion matrix*. In this matrix we can see how the model has difficulties in recognizing the *piano* instrument, and often mistakes it as other musical instruments, probably due to the musical instrument being used in very different contexts in the dataset. Moreover, the model often classify the *organ* class in place of the *piano* class, the *electric guitar* class and the *acoustic guitar* class, as the *organ* is commonly used to mimic the timbre of these instruments. The model effectively classifies the *acoustic guitar* class, and it is able to distinguish it from the *electric guitar* class. It is also effectively distinguishes the *trumpet* and the *saxophone*.

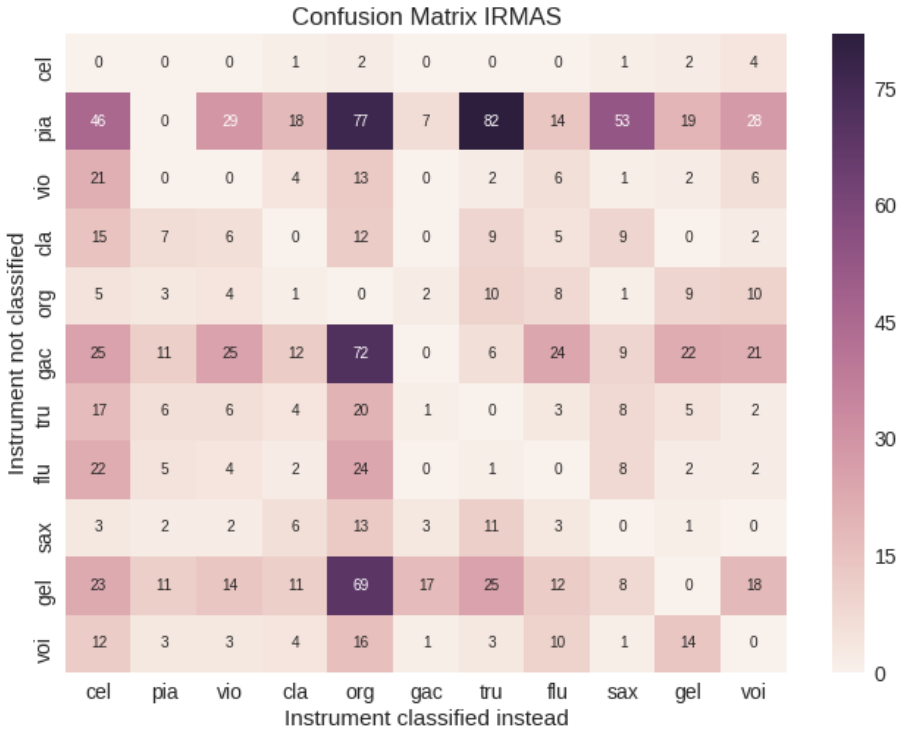


Figure 5.12: Modified Confusion Matrix for the IRMAS dataset.

5.4 Results on MedleyDB + Mixing Secrets dataset

In order to further evaluate our model, we discuss how it performed on the same dataset investigated by Gururani et al. [31]. In this case, they experiment on a mix of different datasets. They used the MedleyDB dataset [5], with the addition of the Mixing Secrets dataset [30] and some additional pop music multi-tracks not contained in any official datasets. The audio excerpt, in this case, are complete songs. They decided to separate the complete data into a training set of 361 tracks and a testing set of 100 tracks. These tracks are labelled with time-resolution of 0.0464 seconds. The training set is augmented using pitch-shifting. The pitch was changed from 6 semitones lower to 5 semitones upper than the original pitch, with increments of 1 semitone. The dataset is labelled for 18 musical instruments: drums (dru), bass guitar (bgtr), pianoforte (pf), male singer (ms), clean electric guitar (cgtr), female singer (fs), acoustic guitar (agtr), distorted guitar (dgtr), violin (vn), cello (vc), synthesizer (syn), voice (vox), double bass (db), flute (fl), tabla (tab), string section (str), electric organ (eorg), electric pianoforte (epf). Every frame has a label for each musical instrument, so we can formulate the problem in a *multi-class multi-label* fashion.

In order to properly use the audio excerpts as inputs, we decided to take the songs and cut them into segments of 10 seconds, as the models used on the OpenMIC-2018 dataset. We evaluated our performance against the results of a model by Gururani et al. [31] which has the same temporal resolution. So, the input matrix for this architecture is of dimensions $128 \times N_S$ with $N_S = 10$, and the output will be a vector of *class probabilities* $\hat{\mathbf{p}}$ of $N_M = 18$ elements. The architecture is defined with 4096 hidden units for the first and the second dense layer, and we applied a *dropout* of 50%. The third dense layer uses a *sigmoid* activation function, since every musical instrument can be. The *autopool* layer has a regularization rate $\lambda = 0.01$ For the training procedure we employed the *binary cross-entropy* as loss function. We used the Stochastic Gradient Descent as optimizer, with an initial learning rate of 0.1. The model was trained on *batches* of 512 samples. The 10% of the training set was used as validation set. When the validation loss stops to decrease, the learning rate was reduced of a factor 0.1. When the validation accuracy stops to increase, we halted the training procedure. In order to evaluate the results, we threshold $\hat{\mathbf{p}}$ with a threshold $\theta = 0.5$, to get binary level. We can then can compare the thresholded predictions against the *ground truth*.

In table 5.3 we summarize the averages results for *precision*, *recall* and F_1 score. The proposed model achieved a *micro* average F_1 score of 65% and

	Micro			Macro		
Model	Precision	Recall	F1	Precision	Recall	F1
Gururani et al.	0.570	0.714	0.634	0.396	0.452	0.388
Proposed	0.680	0.631	0.655	0.511	0.412	0.428

Table 5.3: Comparison of results on MedleyDB + Mixing Secrets Dataset.

Instrument	#Train	#Test	Instrument	#Train	#Test
drum set	300	79	female singer	79	23
electric bass	253	62	string section	24	10
male singer	200	62	elec. piano	24	14
dist. elec. guitar	171	40	elec. organ	22	11
clean elec. guitar	119	34	double bass	21	9
synthesizer	118	33	cello	13	9
acoustic guitar	91	25	violin	10	15
piano	89	24	tabla	9	3
vocalists	84	12	flute	7	7

Table 5.4: Number of training and test set songs, yet to be divided into 10s segments, for every musical instrument.

a *macro* average of 42%. Gururani et al. model, which is a Convolutional Recurrent Neural Network, achieved lower F_1 scores. We can see a big improvement in the *macro* case. This is because their model failing to provide any prediction for some classes with low samples. We can see the number of songs, yet to be divided into 10 seconds segments, which contains the musical instruments in table 5.4. We can see how the low number of samples are correlated with the low F_1 scores for some instruments in figure 5.13. Their model have no predicted samples for the classes *string section* and *electric organ*. Our model works better for the majority of the classes, but under-performs for the *tabla* class, due to the low number of samples, and the *pianoforte* class, probably for the variance of the performances. We list the results for each musical instrument in Appendix B.

In figure 5.14 and figure 5.15 we show, respectively, the *modified confusion matrix* related to our model and the *modified confusion matrix* computed on the test results of Gururani et al. [31]. Their model mistakes the *female singer* class and the *double bass* class for the *bass guitar* class. Our model makes the same error for the *double bass* and the *bass guitar*, but with fewer mistakes, and the reason is that the timbres of the instruments are very similar. Our model appears to make less mistakes on the classes

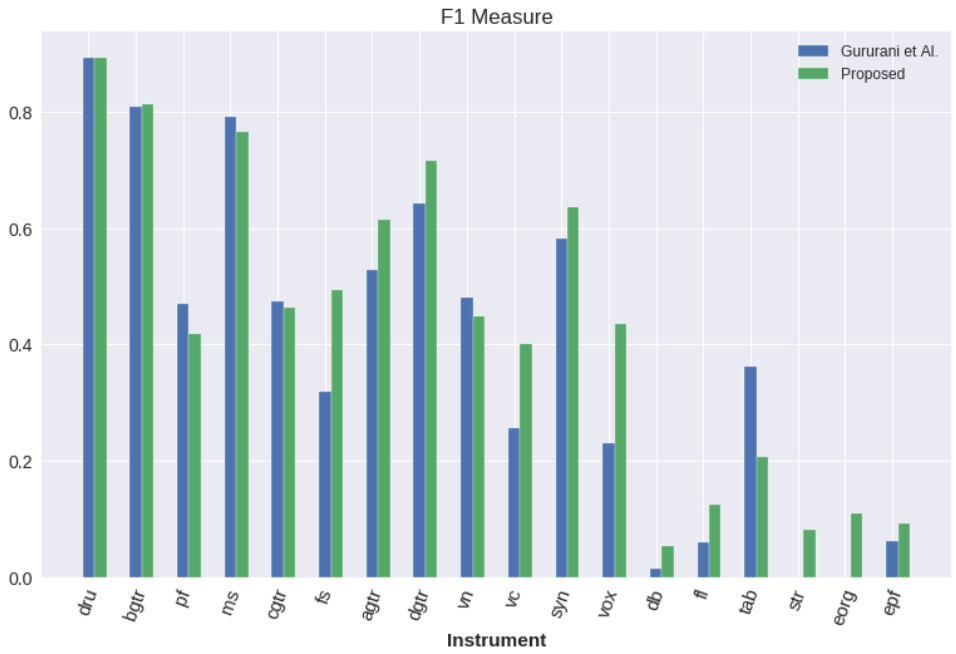


Figure 5.13: Comparison of F_1 scores on Matrix for MedleyDB + Mixing Secrets Dataset, for each musical instrument.

male singer and *female singer*, as it probably better separates the range of frequencies. Generally, our model has problems for the *clean electric guitar* class, which is often mistaken for multiple musical instruments, such as the *pianoforte* and the *electric pianoforte* class, and the *double bass* class, and it is probably due to the variance of the performances on the instrument. The most recurrent mistake for our model is the confusion of the *clean electric guitar* class for the *distorted guitar* class, which is reasonable as they sound similar.

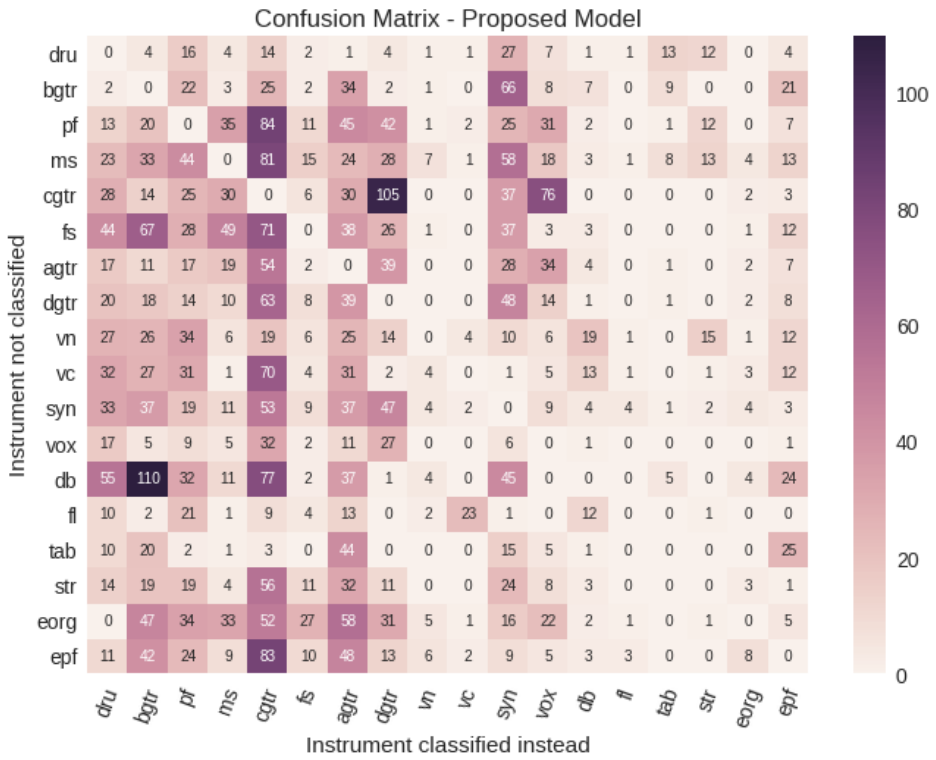


Figure 5.14: Our Modified Confusion Matrix for the MedleyDB + Mixing Secrets Dataset dataset.

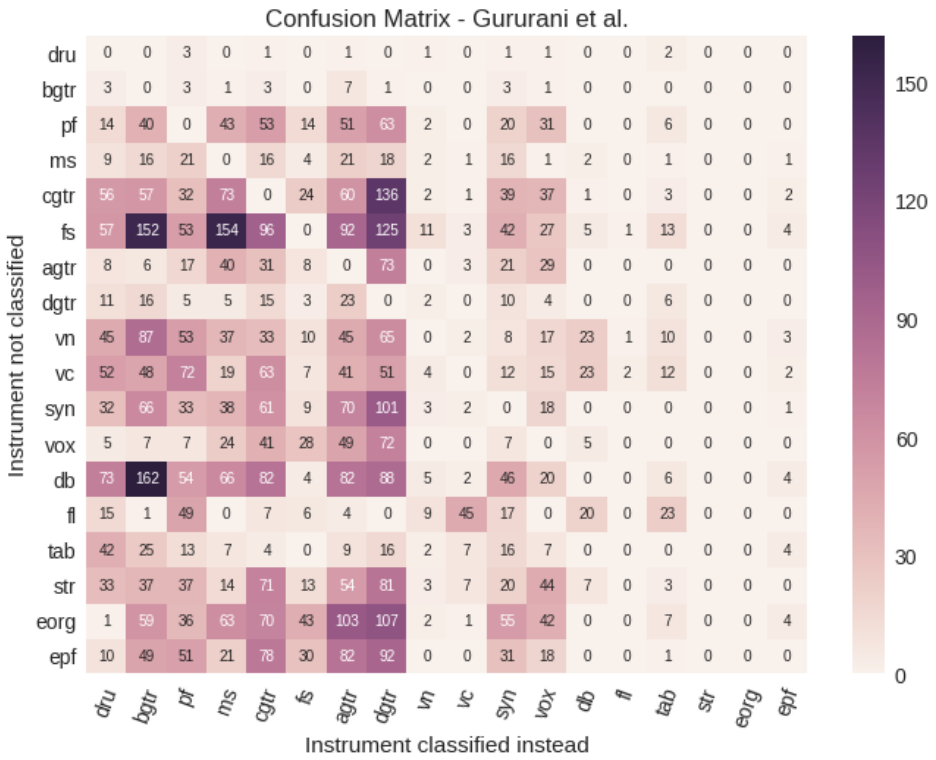


Figure 5.15: Gururani et al. [31] Modified Confusion Matrix for the MedleyDB + Mixing Secrets Dataset dataset.

Chapter 6

Conclusions

In this thesis we proposed an approach to automatic musical instrument recognition based on *transfer learning*. We employed a deep learning approach, using the features extracted from the VGGish [36] model, which are fed to our classifier.

Our work has the purpose of exploiting the knowledge learned from other available, extensive datasets and use it in order to efficiently learn the representation for musical instrument. To do this, we exploited the model trained by Google, the VGGish [36] model, trained on a preliminary version of the YouTube-8M dataset. The features extracted from this model are suitably crafted for the Sound Event Detection task. In this work we showed how the knowledge is easily transferable to the *automatic musical instrument recognition* task. In fact, we showed how a model trained for the detection of sound events of all sorts, such as speech, gunshots, cries, machinery sounds, can be easily adapted to the specific task of recognizing and differentiating the timbre of musical instruments, even between musical instruments which belong to the same family, such as string instruments or wind instruments.

We trained the model on the newly released, large and diverse OpenMIC-2018 dataset [38]. As all the data is labelled in a *weakly* fashion, which means a label is positive if and only if the musical instrument is active in a portion of the clip, we developed our model to output a single clip-level prediction pooled from *time-varying* predictions. To do so, we used a combination of *feedforward* networks, parallelized over the time length of the input audio excerpt. We outperformed the baseline of the dataset developed on the binary classification of each instrument, achieving an average F_1 score of 76%, whereas the baseline classification reached an average score of 70%. We also wanted to fully explore a polyphonic setting where each musical instrument could potentially be active at the same time, as it is much closer

to the real-life case scenario. Unfortunately, the dataset does not have a complete set of labels for each musical instrument on each clip. This would have resulted on the inability of properly formulating a *multi-class multi-label* problem. To tackle this problem, we set up a web survey in order to collect complete annotations for a subset of the dataset. With the help of 68 users, we managed to label 1000 audio clips twice.

For the evaluation, we avoided to use the *accuracy* metric, commonly used for classification tests, as it would report results with low validity in case of highly skewed datasets such as the ones we used. So, we employed the F_1 measure to assess the validity of the trained models, and we achieved satisfactory results. Dividing the aggregation of the collected data in 2 approaches, we achieved a micro average F_1 score of 82% for the *union* approach and a micro average of 81% for the *intersection* approach. Moreover, we employed a *modified confusion matrix* in order to check if the model has the tendency to mistakes certain musical instrument for others.

Finally, in order to test the robustness of the model, we decided to train it on other datasets and evaluate it against other state-of-the-art models. We decided to train on 2 different dataset, the IRMAS dataset [6], which has the training set labelled for the *predominant instrument*, and a mix of dataset based on the MedleyDB dataset [5], with the addition of the Mixing Secrets dataset [30] and some additional pop music multi-tracks not contained in any official dataset. We managed to slightly outperform the state-of-the-art methods with the proposed model, achieving a *micro* average F_1 score of 54% in the first case and a *micro* average score of 65% in the second case.

Another advantage of the work is that most of the pre-processing can be performed only once. In this way, the training procedure will adjust only the weights of the classifier, which will achieve the convergence of the training procedure in a short time, relatively to how much time is needed for the standard *convolutional neural network* to complete the training procedure.

As future improvements, we will integrate the knowledge extracted with the proposed model in order to enhance the performance of systems working on related tasks, such as source separation, automatic playlist generation or music segmentation. For the source separation task, the knowledge of the active instruments can be helpful to separate the tracks in case of a mix of multiple instruments. For the playlist generation task, the musical instruments can be used as a metric of similarity to find similar songs. For the music segmentation task, we can assume that in certain genres of music the activity of the instrument define the different sections of the composition, and therefore help to identify them.

Moreover, we will use the extracted informations to enrich the struc-

tured data of the Music Ontology, and possibly enhance the information connected to the music production process. This will let us better analyse the structure of the composed music and the distribution of the employed musical instruments.

Lastly, we will expand the survey by either annotating more songs or making more annotations on the same subset of 1000 songs, in order to make those results more robust, or providing more data on which a multi-label classifier could be built.

Appendix A

Scores on binary classification

Here we show the F_1 scores obtained for each model built for one specific instrument only.

Instrument	F_1 score on baseline	F_1 score on the proposed model
accordion	0.45	0.51
banjo	0.59	0.61
bass	0.61	0.65
cello	0.78	0.81
clarinet	0.16	0.45
cymbals	0.96	0.95
drums	0.93	0.93
flute	0.46	0.57
guitar	0.98	0.97
mallet percussion	0.73	0.77
mandolin	0.58	0.64
organ	0.36	0.48
piano	0.96	0.96
saxophone	0.83	0.83
synthesizer	0.97	0.96
trombone	0.63	0.71
trumpet	0.69	0.72
ukulele	0.60	0.65
violin	0.89	0.88
voice	0.94	0.95

Scores on Classification with masked input

Here we show the F_1 scores obtained for each instrument with the model built with masked input.

Instrument	F_1 score on baseline	F_1 score on the proposed model
accordion	0.45	0.50
banjo	0.59	0.63
bass	0.61	0.70
cello	0.78	0.80
clarinet	0.16	0.41
cymbals	0.96	0.95
drums	0.93	0.94
flute	0.46	0.60
guitar	0.98	0.98
mallet percussion	0.73	0.74
mandolin	0.58	0.65
organ	0.36	0.61
piano	0.96	0.96
saxophone	0.83	0.82
synthesizer	0.97	0.96
trombone	0.63	0.69
trumpet	0.69	0.74
ukulele	0.60	0.66
violin	0.89	0.88
voice	0.94	0.94

Scores on survey results

Here we show the F_1 scores obtained for each instrument with the models built on the union and on the intersection of the data collected from the survey.

Instrument	F_1 score on *union*	F_1 score on *intersection*
accordion	0.70	0.80
banjo	0.70	0.75
bass	0.78	0.79
cello	0.83	0.80
clarinet	0.8	0.91
cymbals	0.86	0.78
drums	0.88	0.89
flute	0.68	0.82
guitar	0.84	0.83
mallet percussion	0.76	0.84
mandolin	0.74	0.75
organ	0.57	0.89
piano	0.81	0.86
saxophone	0.86	0.75
synthesizer	0.78	0.77
trombone	0.84	0.75
trumpet	0.83	0.78
ukulele	0.83	0.82
violin	0.85	0.80
voice	0.87	0.94

Appendix B

Scores on the IRMAS dataset

Here we show the F_1 scores obtained for each instrument of the IRMAS dataset.

Instrument	F_1 score of the proposed model
cello	0.27
piano	0.57
violin	0.54
clarinet	0.23
organ	0.44
acoustic guitar	0.56
trumpet	0.49
flute	0.57
saxophone	0.72
electric guitar	0.60
voice	0.90

Scores on the MedleyDB+Mixing Secrets dataset

Here we show the F_1 scores obtained for each instrument of the the MedleyDB + Mixing Secret dataset, comparing ours to the results of Gururani et al. [31].

Instrument	F_1 proposed model	F_1 Gururani et al. model
drums	0.89	0.89
bass guitar	0.81	0.81
pianoforte	0.42	0.47
male singer	0.77	0.79
classic guitar	0.46	0.48
female singer	0.49	0.32
acoustic guitar	0.61	0.52
distorted guitar	0.72	0.64
violin	0.45	0.48
cello	0.40	0.26
synthesizer	0.63	0.58
voice	0.43	0.23
double bass	0.05	0.01
flute	0.13	0.05
tabla	0.21	0.36
string section	0.08	0.00
electric organ	0.11	0.00
electric pianoforte	0.09	0.06

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

[2] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *CoRR*, 2016.

[3] Giuseppe Bandiera, Oriol Romani Picas, Hiroshi Tokuda, Wataru Hariya, Koji Oishi, and Xavier Serra. Good-sounds. org: A framework to explore goodness in instrumental sounds. In *ISMIR Proceedings of the 17th International Society for Music Information Retrieval Conference*. International Society for Music Information Retrieval (ISMIR), 2016.

[4] Jayme Garcia Arnal Barbedo and George Tzanetakis. Musical instrument classification using individual partials. *IEEE Transactions on Audio, Speech, and Language Processing (ICASSP)*, 19:111–122, 2011.

[5] Rachel M Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello. Medleydb: A multitrack dataset for annotation-intensive mir research. In *International Society for Music Information Retrieval (ISMIR)*, volume 14, pages 155–160, 2014.

[6] Juan J Bosch, Jordi Janer, Ferdinand Fuhrmann, and Perfecto Herrera. A comparison of sound segregation techniques for predominant instrument recognition in musical audio signals. In *International Society for Music Information Retrieval (ISMIR)*, pages 559–564, 2012.

[7] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT*, pages 177–186. Springer, 2010.

[8] Juan José Burred, Axel Robel, and Thomas Sikora. Dynamic spectral envelope modeling for timbre analysis of musical instrument sounds. *IEEE Transactions on Audio, Speech, and Language Processing*, 18:663–674, 2010.

[9] Emre Cakir, Toni Heittola, Heikki Huttunen, and Tuomas Virtanen. Polyphonic sound event detection using multi label deep neural networks. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2015.

[10] Emre Cakır, Giambattista Parascandolo, Toni Heittola, Heikki Huttunen, and Tuomas Virtanen. Convolutional recurrent neural networks for polyphonic sound event detection. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (ICASSP)*, 25:1291–1303, 2017.

[11] Selina Chu, Shrikanth Narayanan, C-C Jay Kuo, and Maja J Mataric. Where am i? scene recognition for mobile robots using audio features. In *IEEE International conference on multimedia and expo*, pages 885–888. IEEE, 2006.

[12] Arshia Cont, Shlomo Dubnov, and David Wessel. Realtime multiple-pitch and multiple-instrument recognition for music signals using sparse non-negative constraints. In *Proceedings of Digital Audio Effects Conference (DAFx)*. Bordeaux, 2007.

[13] Diana Deutsch. Music recognition. *Psychological Review, University of California, San Diego*, 76:300 – 307, 1969.

[14] Sander Dieleman, Philémon Brakel, and Benjamin Schrauwen. Audio-based music classification with a pretrained convolutional network. In *12th International Society for Music Information Retrieval Conference (ISMIR)*, pages 669–674. University of Miami, 2011.

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

[16] Laura Dilley and Aniruddh D. Patel. Music, language, and the brain Oxford University. *Phonology*, 26:535 – 540, 2009.

[17] Jana Eggink and Guy J Brown. Instrument recognition in accompanied sonatas and concertos. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 4, pages iv–iv. IEEE, 2004.

[18] Antti Eronen. Comparison of features for musical instrument recognition. In *Proceedings of the IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics*, pages 19–22. IEEE, 2001.

[19] Slim Essid, Gaël Richard, and Bertrand David. Musical instrument recognition on solo performances. In *12th European Signal Processing Conference*, pages 1289–1292, 2004.

[20] Slim Essid, Gaël Richard, and Bertrand David. Instrument recognition in polyphonic music based on automatic taxonomies. *IEEE Transactions on Audio, Speech, and Language Processing*, 14:68–80, 2006.

[21] Slim Essid, Gaël Richard, and Bertrand David. Musical instrument recognition by pairwise classification strategies. *IEEE Transactions on Audio, Speech, and Language Processing (ICASSP)*, 14:1401–1412, 2006.

[22] Max Ferguson, Ronay Ak, Yung-Tsun Tina Lee, and Kincho H Law. Automatic localization of casting defects with convolutional neural networks. In *IEEE International Conference on Big Data*, pages 1726–1735. IEEE, 2017.

[23] Neville H Fletcher and Thomas D Rossing. *The physics of musical instruments*. Springer Science & Business Media, 2012.

[24] Ferdinand Fuhrmann. *Automatic musical instrument recognition from polyphonic music audio signals*. PhD thesis, Universitat Pompeu Fabra, 2012.

[25] Ferdinand Fuhrmann and Perfecto Herrera. Polyphonic instrument recognition for exploring semantic similarities in music. In *Proc. of 13th Int. Conference on Digital Audio Effects DAFx10*, pages 1–8, 2010.

[26] Jort F Gemmeke, Daniel PW Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R Channing Moore, Manoj Plakal, and Marvin Ritter. Audio set: An ontology and human-labeled dataset for audio events. In *Acoustics, Speech and Signal Processing (ICASSP), IEEE International Conference on*, pages 776–780. IEEE, 2017.

[27] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning, Boston*. MIT press, 2016.

[29] Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. Rwc music database: Music genre database and musical instrument sound database. In *Johns Hopkins University*, 2003.

[30] Siddharth Gururani and Alexander Lerch. Mixing secrets: a multi-track dataset for instrument recognition in polyphonic music. *International Society for Music Information Retrieval (ISMIR)*, 2017.

[31] Siddharth Gururani, Cameron Summers, and Alexander Lerch. Instrument activity detection in polyphonic music using deep neural networks. *International Society for Music Information Retrieval (ISMIR)*, 2018.

[32] Yoonchang Han, Jaehun Kim, Kyogu Lee, Yoonchang Han, Jaehun Kim, and Kyogu Lee. Deep convolutional neural networks for predominant instrument recognition in polyphonic music. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 25:208–221, 2017.

[33] Yoonchang Han, Subin Lee, Juhan Nam, and Kyogu Lee. Sparse feature learning for instrument identification: Effects of sampling and pooling methods. *The Journal of the Acoustical Society of America*, 139:2290–2298, 2016.

[34] Aki Harma, Martin F McKinney, and Janto Skowronek. Automatic surveillance of the acoustic activity in our living environment. In *IEEE International Conference on Multimedia and Expo*. IEEE, 2005.

[35] Toni Heittola, Anssi Klapuri, and Tuomas Virtanen. Musical instrument recognition in polyphonic audio using source-filter model for sound separation. In *International Society for Music Information Retrieval (ISMIR)*, pages 327–332, 2009.

[36] Shawn Hershey, Sourish Chaudhuri, Daniel P. W. Ellis, Jort F. Gemmeke, Aren Jansen, Channing Moore, Manoj Plakal, Devin Platt, Rif A. Saurous, Bryan Seybold, Malcolm Slaney, Ron Weiss, and Kevin Wilson. Cnn architectures for large-scale audio classification. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017.

[37] Po-Sen Huang, Minje Kim, Mark Hasegawa-Johnson, and Paris Smaragdis. Joint optimization of masks and deep recurrent neural networks for monaural source separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing (ICASSP)*, 23:2136–2147, 2015.

[38] Eric Humphrey, Simon Durand, and Brian McFee. Openmic-2018: an open dataset for multiple instrument recognition. In *International Society for Music Information Retrieval (ISMIR)*, 2018.

[39] Cyril Joder, Slim Essid, and Gaël Richard. Temporal integration for audio classification with application to musical instrument classification. *IEEE Transactions on Audio, Speech, and Language Processing (ICASSP)*, 17:174–186, 2009.

[40] I Kaminsky and Andrzej Materka. Automatic source identification of monophonic musical instrument sounds. *Proc. 1995 IEEE Internetional Conference Neural Networks*, 1:189 – 194, 1995.

[41] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations ICLR*, 2015.

[42] Tetsuro Kitahara, Masataka Goto, Kazunori Komatani, Tetsuya Ogata, and Hiroshi G Okuno. Instrument identification in polyphonic music: Feature weighting to minimize influence of sound overlaps. *EURASIP Journal on Applied Signal Processing*, 2007:155–155, 2007.

[43] Qiuqiang Kong, Yong Xu, Wenwu Wang, and Mark D Plumbley. Audio set classification with attention model: A probabilistic perspective. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 316–320. IEEE, 2018.

[44] A.G. Krishna and Thippur Sreenivas. Music instrument recognition: from isolated notes to solo phrases. *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88.*, 4:iv–265, 2004.

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[46] David Little and Bryan Pardo. Learning musical instruments from mixtures of audio with weak labels. In *International Society for Music Information Retrieval (ISMIR)*, volume 8, pages 127–132, 2008.

[47] Keith Dana Martin. *Sound-source Recognition: A theory and Computational Model*. PhD thesis, Massachusetts Institute of Technology, Boston, USA, 1999.

[48] Nobuo Masataka. The origins of language and the evolution of music: A comparative perspective. *Physics of Life Reviews*, 6:11–22, 2009.

[49] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[50] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th Pyhton In Science Conference*, 2015.

[51] Brian McFee, Justin Salamon, and Juan Pablo Bello. Adaptive pooling operators for weakly labeled sound event detection. *CoRR*, 2018.

[52] Sushobhan Nayak and Ankit Bhutani. Music genre classification using ga-induced minimal feature-set. In *Third National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics*, pages 33–36. IEEE, 2011.

[53] 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 (EUSIPCO)*, pages 2744–2748. IEEE, 2017.

[54] Yves Raimond, Samer Abdallah, Mark Sandler, and Frederick Giasson. The music ontology. In *Proceedings of the 8th International Conference on Music Information Retrieval, (ISMIR)*, 2007.

[55] Jan Schlüter and Sebastian Böck. Improved musical onset detection with convolutional neural networks. In *2014 ieee international conference on acoustics, speech and signal processing (ICASSP)*, pages 6979–6983. IEEE, 2014.

[56] Christian Simmermacher, Da Deng, and Stephen Cranefield. Feature analysis and classification of classical musical instruments: An empirical study. In *Industrial Conference on Data Mining*, pages 444–458. Springer, 2006.

[57] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR*, 2015.

[58] Stanley Smith Stevens, John Volkmann, and Edwin B Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8:185–190, 1937.

[59] Yun Wang, Juncheng Li, and Florian Metze. A comparison of five multiple instance learning pooling functions for sound event detection with weak labeling. *CoRR*, 2018.

[60] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[61] Changsong Yu, Karim Said Barsim, Qiuqiang Kong, and Bin Yang. Multi-level attention model for weakly supervised audio classification. *CoRR*, 2018.

[62] Guoshen Yu and Jean-Jacques Slotine. Audio classification from time-frequency texture. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1677–1680. IEEE, 2009.

[63] Li-Fan Yu, Li Su, and Yi-Hsuan Yang. Sparse cepstral codes and power scale for instrument identification. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7460–7464. IEEE, 2014.

[64] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[65] Dongqing Zhang and Dan Ellis. Detecting sound events in basketball video archive. *Dept. Electronic Eng., Columbia Univ., New York*, 2001.

[66] Xinquan Zhou and Alexander Lerch. Chord detection using deep learning. In *Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR)*, volume 53, 2015.