A Framework for Synthesis of Musical Training Examples for Polyphonic Instrument Recognition

by

Rameel Sethi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Statistical Machine Learning

Department of Computing Science

University of Alberta

Abstract

Music information retrieval (MIR), an interdisciplinary field involving the classifying or detection of structure in music, is essential for processing, indexing, querying and making recommendations from the vast amount of musical data available on the web and in audio library collections. Deep neural networks have yielded state-of-the-art results in several MIR tasks, but are often limited by their reliance on the availability of large amounts of annotated training data. Thus, applying deep learning in MIR may prove difficult when applied to music databases with limited amounts of labelling.

This thesis addresses the question of whether algorithmically compositions generated from a specification of instruments and note events may serve as a viable alternative to real labeled music recordings for use as training data in MIR classification tasks. We propose a simple framework for generation of synthetic musical compositions for use as training data using the popular Musical Instrument Digital Interface (MIDI) protocol, which may be rendered as audio using commonly available synthesizers. In addition, we apply a variety of audio transformations to the generated audio samples for data augmentation purposes. We apply this music synthesis algorithm to the MIR tasks of polyphony estimation (number of instruments sounding) and instrument recognition (which instruments are playing) in polyphonic tracks where multiple instruments may sound simultaneously in each analysis frame, and evaluate our framework on publicly available annotated music datasets. We empirically demonstrate that pure synthesis of a musical training set without usage of a training set of music yields statistically significant improvements over a random or majority classifier. The main contribution of this thesis is to show that synthetic musical composition generation coupled with data augmentation has the potential to aid content-based MIR in music collections with limited amounts of annotation.

Preface

Chapter 5 is an adaptation of "Training Deep Convolutional Networks with Unlimited Synthesis of Musical Examples for Multiple Instrument Recognition", in proceedings of the Sound and Music Computing Conference 2018. This paper contains contributions from the following collaborators:

- Noah Weninger with initial code for the synthesis loop
- Dr. Abram Hindle with code for the noising framework and manuscript editing
- Dr. Vadim Bulitko for manuscript editing
- Dr. Michael Frishkopf for manuscript editing

Dedicated to Nadira Khan (1932-2017)

Acknowledgements

I would like to express my utmost gratitude to my supervisors Dr. Abram Hindle and Dr. Vadim Bulitko for their constant guidance throughout my degree. Dr. Hindle tirelessly guided me through every step of the process, from learning how to conduct research to manuscript editing. Dr. Bulitko always provided a new angle for approaching problems, and supplied constructive criticism which has vastly improved my research and writing skills.

I would also like to thank Dr. Michael Frishkopf for his support during this project by offering insights from a musical perspective, as well as the Deep Learning for Sound Recognition (DLSR) group for bringing people from many different areas together for fruitful discussions.

I thank Gregory Burlet for our discussions regarding polyphonic transcription, and Noah Weninger for his initial contributions towards the project.

I thank the members of EMPLAB, including Eddie Antonio Santos, Hazel Campbell, Shaiful Chowdhury, Candy Pang and Stephen Romansky for their kindness and helpful advice. I also thank the members of the AGI lab, including Sergio Poo Hernandez and Devon Sigurdson, for their discussions and advice.

Special thanks are owed to KIAS and the University of Alberta for their generous financial support without which I could not complete this work. I also thank the NVIDIA Corporation.

Additionally, I thank Dr. Hani Henein of the Department of Chemical and Materials Engineering at the University of Alberta for his kindness and taking time out of his busy schedule for a cup of coffee. Raja Junaid Jahangir deserves special thanks for being a wonderfully supportive housemate and putting up with me for most of his Masters degree. I also would like to thank Kyle Fisher, Clyde Gross, Michael Kelly, Daniel Kaufmann and Jason Melendez for their friendship and support both during undergraduate and graduate school.

Finally, I thank my parents, Haroon and Raiyan, my sisters, Rishma and Raniya, and my grandfather, Abdul Jabbar, for their unwavering love and support.

Table of Contents

List of Tables

List of Figures

List of Acronyms

CNN convolutional neural network

DFT discrete Fourier transform

MFCC mel-frequency cepstral coefficients

MIDI Musical Instrument Digital Interface

MIR music information retrieval

MIREX Music Information Retrieval eXchange

STFT short-time Fourier transform

ZeroR majority class classifier

Chapter 1

Introduction

The rise of the Internet as a platform for the consumption and sharing of audio, among other forms of multimedia, has been accompanied by a sharp increase in the amount of music available online. This music may be available on audio streaming platforms as well as in digital archives.

With the introduction of streaming sites like Spotify [36] and Shazam [82], millions of tracks have become publicly available for listening. Services like Shazam allow users to find a track simply by humming melody; the platform services then proceed to recommend tracks similar to the recorded user input. Spotify suggests songs similar to ones previously listened to using recommendation systems based on user listening history and genre preferences.

These systems rely heavily on access to large-scale music databases for training machine learning models around which their recommender system infrastructure is architected, either using audio metadata or content [57]. Once such datastores are available, *music information retrieval* (MIR) techniques play a critical role in efficiently indexing and querying the information contained. Machine learning approaches have recently achieved state-of-the-art performance in several MIR tasks, compared to more traditional techniques such as signal processing [5], [67], [44].

Content-based music information retrieval [19] focuses on using a suitable representation of the audio content, rather than metadata, of musical recordings to classify the recordings according to a predefined set of labels. In both cases, tasks such as chord, genre, instrument and mood recognition may be performed. These tasks may be at the track level (e.g. genre recognition), or at the audio analysis frame (i.e., a window of arbitrary size determined by the MIR researcher) level (e.g. chord recognition). Content-based MIR may be used to improve the recommendations given by services

such as the above-mentioned, since certain metadata categories such as genre or artist (according to user listening preferences) may be biased towards certain patterns in content, such as chord progressions.

However, these models are often biased towards Western music [6], and may often not prove useful when applied to non-Western music [64]. Non-Western music differs from Western music in several aspects. It is often studied without notation [78]. Also, the *scales* (i.e., a set of notes which serve as a basis for constructing melodies) used often have different numbers of pitches than the 12-semitone equal tempered scale prevalent in Western music, and [20]. Additionally, some genres such as African music are dominated by rhythmic and percussive devices [54]. Therefore MIR techniques rooted in Western music do not necessarily generalize to non-Western music [11], [42].

Ethnomusicology, the study of music in cultural context, is a field where MIR techniques have given rise to an emerging subfield known as computational ethnomusicology [77], which is part of the larger effort towards algorithmic analysis of music. Progress using machine learning techniques, however, has been rather limited so far largely due to a lack of annotated training data, even though large numbers of field recordings exist in ethnomusicology archives (though not on the scale available in streaming services). Scarcity of training data makes it difficult to apply MIR techniques to ethnomusicological collections. One reason is the presence of copyright restrictions on music, making large-scale MIR difficult outside of academic music datasets and organizations with large amounts of music available [25]. The training data that is available in such archives is often not digitized or lacks even the most basic labelling such as metadata; it is also difficult to collect such data in a machine-readable format since fieldwork in ethnomusicology largely relies on personal interaction in the cultures of interest [21].

Even if large amounts of audio are available for applying *supervised* (i.e., learning from datasets containing labels) content-based MIR techniques, it is very difficult to apply these algorithms when the music is accompanied by little or no annotation. Audio is difficult to annotate manually for a number of reasons. Primarily, audio is a medium that needs to be listened to as a whole before annotations can be made, either at the track level or the timestamp level; when sped up, the human ear may not be able to capture salient characteristics of the track being played. Also, labelling an audio track at the *timestamp* level (i.e., a specific point in time) or the *window* or *frame* level (i.e. a segment of a certain duration within a track which may contain musical properties) is prohibitively expensive, especially when the recording may potentially be of several hours in length and the

audio analysis frame is a fraction of a second [84]. Additionally, even if experts decide to label audio manually, two experts may disagree on the label to be assigned to the same segment of audio depending on how their ears perceive the played sound, which may result in *noisy* or ambiguous audio labels for information such as genre, mood and instrumentation [8]. Also, hiring expert annotators may prove costly. These issues result in audio being a medium that does not lend itself well to expert annotation due to the time-consuming and error-prone nature of the task.

Learning from large collections of audio with limited or no annotations is therefore a challenging task. One possible idea to address the lack of annotated training data is to use algorithmically generated musical training data. For several years, the music community has been producing synthesized music [10] that imitates the properties of physical instruments using a wide array of publicly available hardware and software synthesizers. The motivation for using artificial musical data lies in the fact that using simple algorithmic generation rules, we can generate a practically unlimited amount of training data annotated with ground truth labels.

In this thesis, I explore the question of whether content-based MIR tasks may be performed in audio databases of *real music* (i.e., recordings of excerpts from songs performed using physical instruments) using *synthetic musical training data* (i.e., audio samples generated using a synthesizer rather than played with physical instruments). Specifically, I explore the related tasks of polyphony and instrument recognition in polyphonic music (i.e., more than one instrument playing at a time), and ask whether adequate performance on test songs can be achieved from synthetic training music which may not encode all the information contained in a song recording.

1.1 Contributions

This thesis makes the following contributions:

- We introduce a simple framework for unlimited synthesis of musical compositions for generation of audio as part of a training loop for supervised classifiers. We show that deep convolutional networks trained on audio rendered from these synthetic compositions show overall weak, but still yield better and statistically significantly performance than a ZeroR (majority class) classifier on test datasets of real recorded music without usage of any real training set, suggesting there is potential for synthetic pretraining of deep neural networks as a first stage in MIR tasks.
- We analyze the effect of reducing the search space of random composition generation. We find that generating compositions only using MIDI instrument presets mapped to physical instruments found in the test set results in better performance than random sampling from the entire specification without incorporating knowledge of the test set. This may aid MIR researchers working with music databases in training MIR models according to test datasets of their interest with a pre-defined taxonomy.
- We experiment with a new 2D visual audio representation, which incorporates phase and phase gradient information in addition to the magnitude of the short-time Fourier transform (STFT). These three signals are combined as channels of an RGB image for input to a convolutional neural network. This may open up research into new audio feature engineering methods that can utilize the phase information of signals for MIR.

1.2 Thesis organization

This thesis is organized as follows. Chapter 2 presents necessary background material on music information retrieval, signal processing and machine learning in the context of MIR. Chapter 3 describes our problem formulation. Chapter 4 presents related work for polyphony estimation and instrument recognition. Chapter 5 provides our framework for generation of synthetically composed audio, and subsequently shows that deep convolutional networks trained on music rendered from synthetic MIDI compositions can be applied to instrument recognition in polyphonic music. Chapter 6 proposes one improvement to our framework by generating training audio only according to the

instruments present in the test data. Finally, Chapter 7 concludes this thesis and suggests a few possible extensions to our work.

Chapter 2

Background

In this chapter, we begin with a brief overview of background material on music representations and audio signal processing techniques commonly used in music information retrieval (Section 2.1). Next, we give a brief introduction to music information retrieval (MIR) (Section 2.2). Finally, we survey the application of several machine learning algorithms to MIR, with emphasis on convolutional neural network (CNN)s (Section 2.3).

2.1 Music and Audio Concepts

Music may be represented in several forms including *sheet music*, *symbolic*, and *audio*. Sheet music representations, which represent musical scores either in printed form or as digitized images [52], play an important role in tasks such as transcription; however for the purposes of this thesis we only discuss symbolic and audio representations. It is worth noting that the boundaries between sheet and symbolic music are not precisely defined; we take symbolic music to be a representation of music that may be processed by a computer [52].

2.1.1 Symbolic Representations

The choice of music representation is important for this thesis since we are focusing on synthetic composition of music. Symbolic music representations, while machine-readable, are open to interpretation by the performer; as such, not all information pertaining to a given recording can be captured in symbolic representations. Additionally, instruments may be tuned differently; thus no two

performances of the same song will be identical. There exist several different symbolic representations including MusicXML [22] and piano rolls; however, there is one symbolic representation that is particularly pervasive in the music community and has stood the test of time for its portability across different hardware and software standards, namely Musical Instrument Digital Interface (MIDI).

2.1.2 Notes, Harmony and Chords

Two commonly used terms when describing music compositiions are notes and chords. A *note pitch* is the perceived frequency of a sound being played. To understand chords, we first describe the term *harmony*, which refers to the perception of sounding of different notes simultaneously as "a cohesive entity") [52]. From an MIR perspective, a *chord* is a harmonic set of note pitches usually containing two or more notes sounding simultaneously on either a single instrument, or across multiple instruments.

2.1.3 MIDI

Musical Instrument Digital Interface [50] is a standard protocol for communication of musical messages between hardware (e.g., electronic synthesizers and keyboards) as well as software. Although its original intended used was as a musical communication protocol, MIDI has been widely adopted as a representation of music. Additionally, even though MIDI was originally envisioned as a standard between all devices, there are in fact several MIDI specifications which differ in a variety of aspects such as instrument presets. In this thesis, however, we only consider the General MIDI System Level 1 (GM1) specification.

Figure 2.1 shows an example piano roll MIDI representation for a single instrument. From this representation, we may deduce the note *pitches* (the perceived frequency of the sound being played) being played as well as note onset and offset times.

A MIDI device has 16 *channels*, which are paths over which MIDI messages travel from source to destination. Channel number 10 is reserved for percussive instruments. Each of these channels can be assigned to one of 128 *program numbers*, which denote a certain instrument or sound playing according to a predefined specification. MIDI messages encode note event information including note *onset* and *offset* which specify note timing and duration. The note *velocity* is an integer ranging

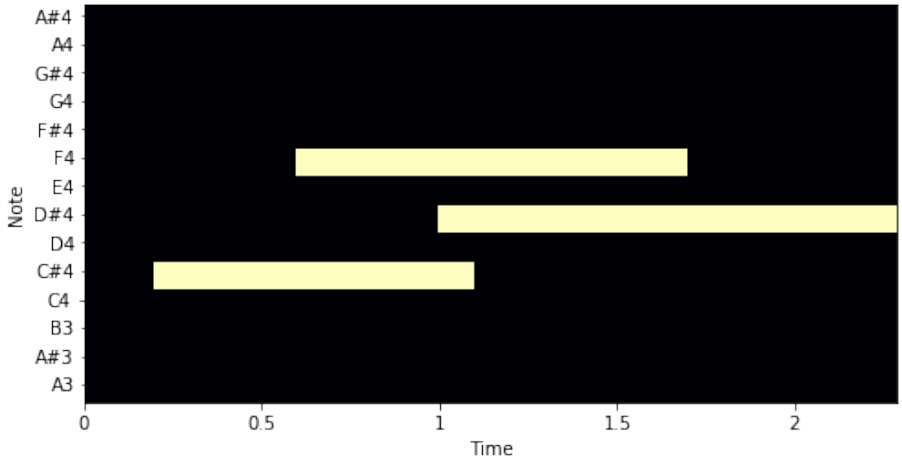


Figure 2.1: A piano roll MIDI representation of three notes with note numbers 60, 64 and 62 respectively.

between 0 and 127 indicating the intensity of the note when initially played. The MIDI *note number* is an integer between 0 and 127 which stores the pitch of the current note being played.

MIDI is somewhat limited as a music representation. One interpretation of this limitation is that MIDI is biased toward representing all instruments and note pitches as key strikes of a pitched keyboard with a certain velocity. This may not accurately represent how different kinds of instruments are played, especially some types of percussion instruments which are not pitched.

2.1.4 Audio Representations

Although *end-to-end learning* (i.e., learning directly from raw digitized audio signals without directly converting the audio to a known representation) has recently been proved effective [15], some preprocessing of the audio signal is more prevalent in the MIR literature. Signal processing techniques allow injection of more prior information about the audio waveform into the desired audio feature representation as input to any machine learning algorithm of interest. In this section, we will give an overview of some common signal processing techniques applied to MIR.

2.1.4.1 Discrete Fourier Transform

The discrete Fourier transform (DFT) is a common signal processing technique for transforming a signal from time to frequency domain. For a given discrete signal x of length N, the DFT X is given by Equation 2.1:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-2\pi i k n / N} \tag{2.1}$$

for $k \in \{0, \ldots, N-1\}$.

On its own, the DFT is not commonly used for signals of finite length, or signals containing a mixture of fundamental frequencies. This may cause leakage [28], which is the appearance of local artifacts at the boundaries of signals. The temporal information of a raw signal is lost upon conversion to the frequency domain. Some method of applying this transform over a windowed version of the signal is required.

2.1.4.2 Short-time Fourier Transform

The short-time Fourier transform (STFT) is a windowed version of the DFT. Equation 2.2 gives the formula for calculating the STFT, X, using a Hamming window [53], which is the most commonly used window function for STFT calculation, although the Hann and Blackwell window functions are also common [53]. For a given discrete signal x of length N, the STFT X is given by Equation 2.2:

$$X(t, k) = \sum_{n=0}^{N-1} w(n) x\Big(n + t\frac{N}{2}\Big) e^{-j2\pi k n / N} \tag{2.2}$$

where N is the length of the input signal x, w is the window function used, and $k \in \{0, \ldots, N\}$.

A representation of the signal which displays the relationship between time and frequency is often useful in music analysis.

2.1.4.3 Spectrogram

A spectrogram is a time-frequency representation of a signal. When visualized, frequency buckets are typically plotted on the vertical axis against time on the horizontal axis. There is a tradeoff between time resolution and frequency resolution, similar to Heisenberg's uncertainty principle [59]. If a finer

time resolution is desired, frequency resolution becomes more coarse and vice versa. The choice of which resolution to make fine-grained largely depends on the signal processing task in question.

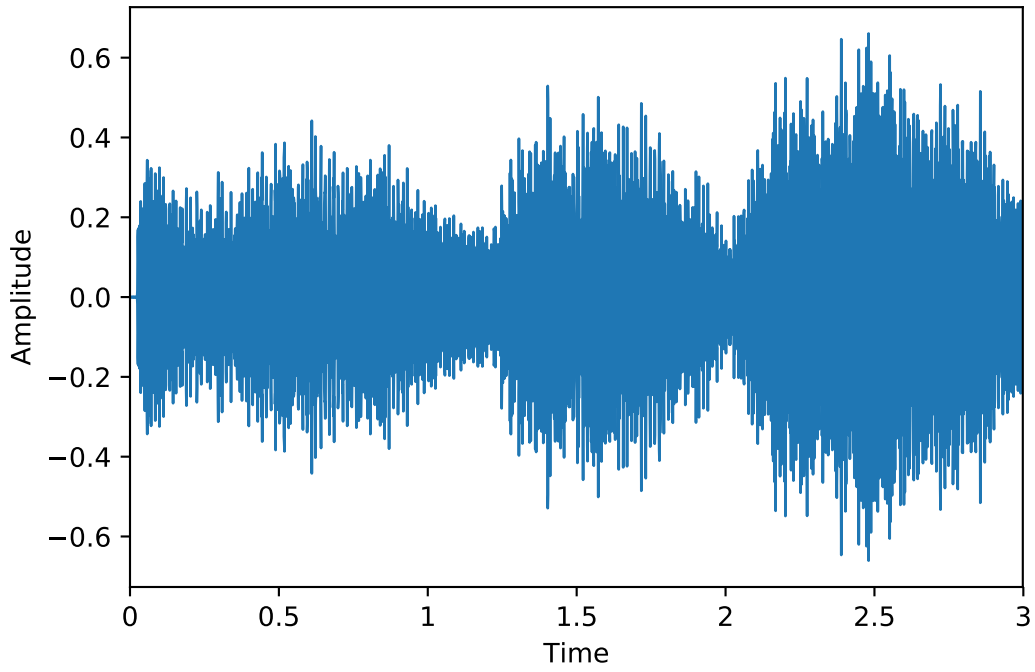
There are several different time-frequency representations used in signal processing for music analysis, but the STFT magnitude spectrogram is the most commonly used. Since the STFT is a complex-valued function, both magnitude and phase information are available, though the phase information is not often used. Figure 2.2 shows an audio waveform for an excerpt from Beethoven's Fifth Symphony and the resulting spectrogram obtained from taking the magnitude of the STFT. Note that the amplitude peaks of the raw audio signal correspond to the increases in intensity (the non-blue regions) of the spectrogram. Also, the frequencies carrying the largest amplitudes (the red regions of the spectrogram) are concentrated in frequencies below 2000 Hz.

2.1.5 The distinction between real and synthesized music

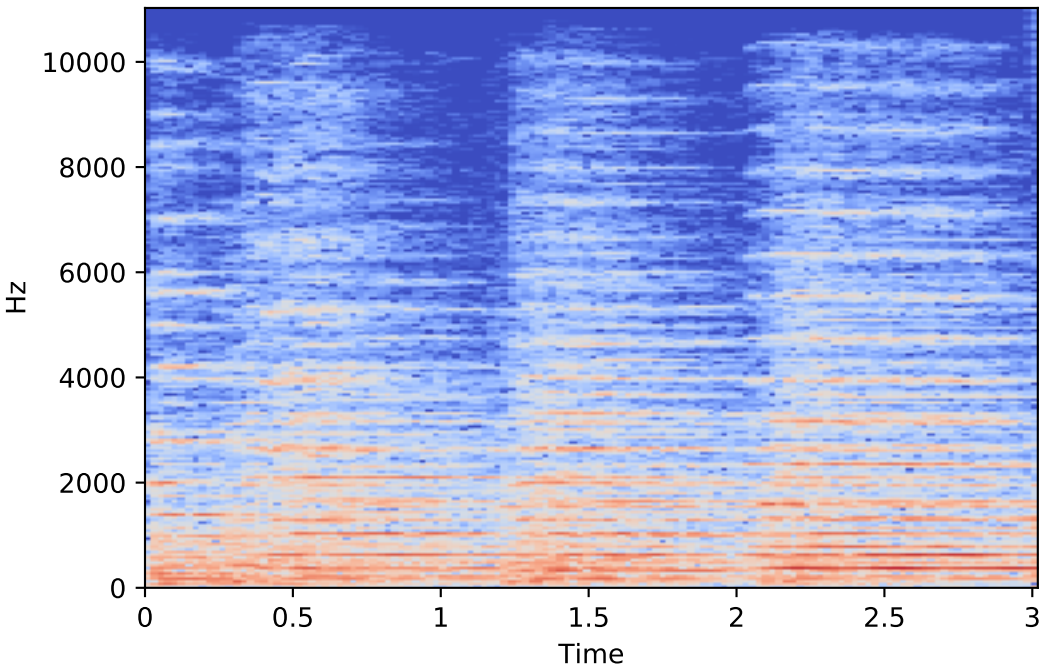
It is not possible to provide a universal definition for the term *music*; the distinction between music and noise is not clear as the kinds of sounds considered as music are defined by individual cultures [56]. In this thesis, we consider real music to be *produced* or *recorded* with *physical* instruments [76]. The term *synthesized* music may refer to any audio that is rendered from a representation that was based on a prior recording of music (e.g., a MIDI file converted from a studio recording) [76]. For the purposes of this thesis, we restrict this definition to audio that is rendered from representations that are *not* based on prior recordings.

2.2 Music Information Retrieval

Music information retrieval (MIR) is a collective term referring to techniques used to classify or determine attributes of music [4]. Like machine learning in general, MIR tasks may be supervised or unsupervised in nature. Supervised learning from music usually involves classification of some form, and may include tasks such as chord, genre or mood recognition. In contrast, unsupervised approaches to music information retrieval may include audio decomposition using methods such as principal component analysis and non-negative matrix factorization, or alignment of audio representations using dynamic time warping. In this thesis, we only focus on classification (i.e., supervised learning from music).



(a) Audio waveform



(b) Spectrogram

Figure 2.2: An audio waveform of the first three seconds of Beethoven's Fifth Symphony and its corresponding STFT magnitude spectrogram.

2.2.1 Audio Content-based MIR

Audio content-based MIR is a key component of music recommendation systems [57]. MIR can be performed at the track level (e.g., genre and mood recognition) or at the audio analysis frame level (e.g., chord and instrument recognition). Note that content-based MIR techniques can be applied either to symbolic or audio representations of music [52].

Some MIR tasks do not use audio content at all. An example is analysis of song lyrics [48]. In this thesis, our focus is limited to content-based MIR which is concerned with tasks relating solely to audio content.

2.3 Machine Learning in MIR

Like computational ethnomusicology, machine learning algorithms have only recently emerged as a trend in MIR. As mentioned in Section 2.2, while both supervised and unsupervised algorithms are commonly employed in MIR, we will only discuss supervised approaches in this thesis.

2.3.1 Feature Extraction

Classical machine learning algorithms rely on feature engineering of hand-crafted representations. Once these representations are computed, a machine learning algorithm may be used to train a classifier for inference on unseen test data. In MIR and, more generally, speech and audio, mel-frequency cepstral coefficients (MFCC) are a popular feature representation for audio [51]. This method computes several coefficients up to a specified number (in practice, the first 12 coefficients are often used [86]) using the Fourier and discrete cosine transforms on a mel scale, which provides a perceptual pitch scale.

Feature engineering is a difficult task and often requires expert domain knowledge, especially in audio. As an alternative, we may try *learning* feature representations directly from raw audio, instead of hand-crafting these representations.

2.3.2 Learning Algorithms

Once a suitable feature representation has been computed, a machine learning algorithm may be used to train a classifier using training data in order to perform inference on unseen test data. The

choice of learning algorithm depends on the type of task (e.g., supervised or unsupervised) as the amount of data and nature of features. There are many available learning algorithms; here we choose to describe only a couple of popular algorithms commonly used in MIR.

The simplest possible classifier is termed majority class classifier (ZeroR), which simply predicts the majority class label for all instances. We use this classifier as a baseline for our experiments in this thesis.

Support vector machines (SVMs) have seen much use in MIR classification tasks due to their ability to learn *nonlinear decision boundaries* (i.e., boundaries between output classes that are more complicated functions than straight lines) with *soft margins* (i.e., some tolerance between class boundaries is permitted, rather than rigid boundaries). SVMs are widely used in a number of MIR tasks including genre recognition [45] and instrument recognition [69]. While SVMs employing the *kernel trick* (i.e., a computation method that enables expression of high-dimensional functions in a low-dimensional space) can learn nonlinear functions, we use deep neural networks instead as they are able to learn nonlinearities more efficiently in the multi-class case, whereas for SVMs several binary classifiers must be combined [2].

Boosting is an approach that has found remarkable success in machine learning competitions [71]. The main idea is to train several weak classifiers and then combine the results to produce a single, robust classifier. Since multiple classifiers are combined, it is also known as *ensemble learning*. In MIR, gradient boosting has achieved state-of-the-art results in predominant instrument recognition [69]. Despite their success, we instead focus our attention on deep neural networks since they require less feature engineering compared to gradient boosting which relies on computation of several handcrafted features from audio.

2.3.3 Neural Networks, Deep Learning and Convolutional Networks

Artificial *neural networks* are structures for learning feature representations from inputs. They are composed of input and output layers of neurons representing input data and outputs, between which one or more *hidden layers* may be present along with interconnections between each layer and the previous layer. Each neuron has an *activation function* which allows learning of non-linear representations. Learning in neural networks occurs via backpropagation [61], which allows gradient updates to be computed in each layer and propagated to previous layers.

The term *deep learning* refers to the stacking of multiple hidden layers of neurons for learning multiple levels of representations [38].

Convolutional neural networks (CNNs) are artificial neural network architectures that replace initial fully connected layers with convolutional layers. These networks, have the ability to learn spatial features at multiple levels [30] and are invariant to image distortion effects such as translation, which often results in improved performance over hand-crafted features and eliminates the need for area experts to spend large amounts of time and effort in finding effective feature representations. The success of deep learning has its origins in areas such as computer vision [31], [39] and speech recognition [29]. In particular, deep convolutional networks such as AlexNet [37] achieved state-of-the-art performance on the ImageNet [14] image dataset classification task, spurring further improvements such as VGG [68] and GoogleNet [70] which added increasing numbers of hidden layers.

The use of deep learning has improved upon prior state-of-the-art in several MIR tasks such as automatic music transcription [33], [67] and music recommender systems [57]. *Deep belief networks* (i.e., deep graphical models which are trained layer-by-layer instead of as a whole) have also been used in conjunction with hidden Markov model frame-smoothing methods for tablature transcription [5]. In the domain of sound event classification, recent developments have proposed learning a feature space for discrimination between classes of sounds by predicting mixing ratios of different classes of sounds [75].

Our aim in this paper is not to investigate novel approaches to improving performance of deep convolutional networks in MIR tasks, but rather to simply utilize existing networks while focusing on synthetic data generation. Therefore, we use off-the-shelf convolutional networks with minor modifications as shown in Chapter 5.

2.3.3.1 Stochastic Gradient Descent and Backpropagation

In general, machine learning algorithms involve some form of *optimization* of an *objective* or *loss function*. Given an objective function $J(x, \theta)$ where x denote the samples presented and θ denote the function parameters, we seek to *minimize* the value of J. *Gradient descent* [7] is one method of performing this minimization, by stepping in the opposite direction of the gradient of J.

Updating the objective J after processing each sample in x is computationally expensive. *Stochastic gradient descent* (SGD) [58] is an improvement to batch gradient descent that computes weight updates after processing each training example, rather than waiting to process all training examples

w: Weight vector
η: Learning rate
Q: Loss function
n: Number of samples **while** $Q(w)$ *is not yet minimized* **do**
  Randomly shuffle examples in training set
  for $i = 1, 2, \ldots, n$ **do**
    $w \leftarrow w - \eta \nabla Q_i(w)$
  end
end

Algorithm 1: Pseudocode describing SGD algorithm

before computing a single weight update. The term *stochastic* refers to the fact that stochasticity (or randomness) in this training algorithm is caused by presenting samples to the algorithm in a random order. Algorithm 1 describes pseudocode for the SGD algorithm.

The SGD algorithm can be demonstrated with a simple example, namely the *exclusive-or* (XOR) function. The XOR function is a binary function on two binary inputs and a binary output that is 1 when exactly one of the inputs is a 1, and 0 otherwise. Given four samples $X = [(0,0),(0,1),(1,0),(1,1)]$ and $y = [0,1,1,0]$, the first iteration of SGD may, for example, randomly shuffle the samples and process them in the order $X = [(1,0),(0,0),(1,1),(1,0)]$ and $y = [1,0,0,1]$; weight updates will be computed on processing each of these samples one at a time. The next iteration may process the samples in the order $X = [(1,1),(1,0),(0,0),(0,1)]$ and $y = [0,1,0,1]$, and so on.

Although XOR is a rather simple function, it cannot be learned by a linear classifier since, given the data distribution for the XOR function, there is no way to separate the output classes with a line. We can learn XOR using a neural network with one hidden layer with eight hidden units as shown in Figure 2.3, with each neuron using an activation function such as the *rectified linear unit* (ReLU) [55] given by $f(x) = max(0, x)$, or the *sigmoid* given by $f(x) = \frac{1}{1+e^{-x}}$.

At the heart of training deep neural networks is a computational technique known as *backpropagation* [83]. There are two steps performed in this computation; a *forward pass* and a *backward pass*. The forward pass involves propagation of an input vector presented at the input layer through each layer of the network, until the output layer is reached. The backward pass compares the output value obtained to the ground truth value and computes an error using a provided loss function. This error is used to compute the gradient of the loss function in terms of the weights of the current

layer, which in turn is used to update these weights using SGD as described above. The error is then propagated this error through weight updates to the previous layers in the network. This cycle repeats until a stopping criterion is achieved.

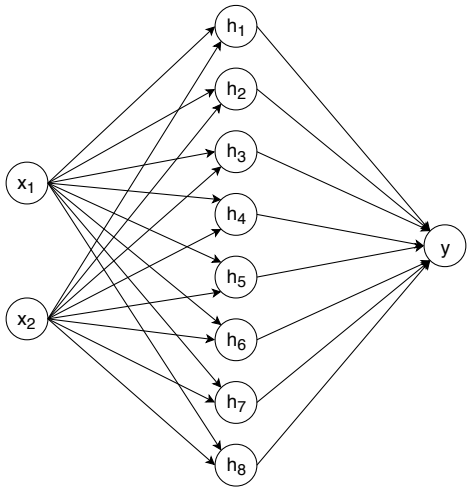


Figure 2.3: Neural network with 2 inputs, 1 output and 1 hidden layer with 8 units

Using a learning rate $\eta = 0.1$ and sigmoid activation functions for the neurons, weight updates for the first two (and the 1000th) *epochs* or passes over the training data, may be computed as shown in Table 2.1. *ypred* is the predicted value of the output neuron; *err* is the *binary cross-entropy* between the ground truth output y and the predicted output $\bar{y}$ ($err = -y \log \bar{y} - (1 - y) \log(1 - \bar{y})$), which is used as a loss function; and w_1 and w_2 are the weights of the hidden and output layers respectively. For the forward pass, the hidden layer outputs h_1 and h_2, and the output y are computed using the sigmoid of a weighted sum of inputs. For the backward pass, weights are updated by computing the gradient of the error at first the output neuron, and then the hidden layer neurons, with respect to the weights. The training data is shuffled between the first and second epoch to introduce stochasticity. Before training, the network weights are randomly initialized.

We can see that by the 1000th epoch, the cross-entropy loss has been minimized and the XOR function has been learned by the network.

In practice, weight updates for SGD are computed over *mini-batches* rather than a single example in order to reduce the amount of noise introduced by only using a single sample for weight updates. The *mini-batch size* (or simply batch size) is a hyperparameter that requires tuning.

Epoch	x_1	x_2	y	$\bar{y}$	err	w_2	w_1
-	-	-	-	-	-	[0.685, 0.137, ..., 0.724]	[-0.415, -0.645, ..., 0.740]
1	0	0	0	0.389	0.380	[0.685, 0.153, ..., 0.747]	[-0.415, -0.645, ..., 0.002]
1	0	1	1	0.384	0.136	[0.685, 0.153, ..., 0.764]	[-0.415, -0.645, ..., 0.062]
1	1	0	1	0.387	0.136	[0.685, 0.153, ..., 0.761]	[-0.415, -0.645, ..., 0.021]
1	1	1	0	0.373	0.767	[0.685, 0.153, ..., 0.684]	[-0.415, -0.645, ..., -0.030]
2	1	0	1	0.389	0.872	[0.685, 0.153, ..., 0.699]	[-0.415, -0.645, ..., 0.009]
2	1	1	0	0.389	0.135	[0.685, 0.153, ..., 0.633]	[-0.415, -0.645, ..., -0.035]
2	0	1	1	0.384	0.135	[0.685, 0.153, ..., 0.656]	[-0.415, -0.645, ..., -0.011]
2	0	0	0	0.394	0.757	[0.685, 0.153, ..., 0.656]	[-0.415, -0.645, ..., -0.011]
1000	1	1	0	0.014	0.001	[0.685, 3.016, ..., 1.920]	[-0.415, -2.231 , ..., -0.009]
1000	1	0	1	0.998	0.003	[0.685, 3.016, ..., 1.920]	[-0.415, -2.231 , ..., -0.009]
1000	0	0	0	0.001	0.001	[0.685, 3.016, ..., 1.920]	[-0.415, -2.231 , ..., -0.009]
1000	0	1	1	0.997	0.005	[0.685, 3.016, ..., 1.920]	[-0.415, -2.231 , ..., -0.009]

Table 2.1: Backpropagation computation using SGD over 1000 epochs (or passes) on training set for XOR

2.3.4 Transfer Learning

Transfer learning is the transfer of learned features between a source task and a target task. The motivation for transfer learning is that there may not be much available labeled training data for a given task. However, if there exists a similar task for which there is a lot of available data, a classifier may first be trained on this source task and then the learned parameters may possibly be useful in making predictions for the target task (optionally with some additional training on data from the target task).

Transfer learning in MIR is still in early stages. However, one recent application makes use of features learned through deep convolutional networks trained on a genre recognition dataset as the source task. When applied to a range of music tagging target tasks, state-of-art results are achieved on several of these tasks [9].

In this thesis, we will not focus on transfer learning between different MIR tasks; however, we investigate an implicit transfer of knowledge which involves learning from algorithmically generated compositions and applying these learned features to real (i.e., recorded) music.

2.3.5 Multi-label Classification

Multi-label classification problems are distinct from multi-class problems in that multiple classes may be predicted per instance as opposed to just one class for multi-class classification, in addition to having multiple classes.

We use the MetaLabeler approach [73] for multi-label instrument classification at the frame level (later aggregated across all frames into an overall track-level classification), since it has proven effective in a related task of polyphonic guitar transcription [5]. MetaLabeler trains a separate classifier to predict label cardinality, and imposes this cardinality as a constraint on the number of output classes predicted by a multi-label classifier.

In our case, multi-label instrument classification is subject to a maximum *polyphony* (number of instruments simultaneously sounding, described in more detail in 3) constraint of 3, since accuracy of polyphony estimation decreases at higher polyphony values due to frequency overlapping of frequency spectra, although in the MIR literature maximum polyphony levels of 5 have been studied [17].

Chapter 3

Problem Formulation

In this chapter we formally define the twofold problem of polyphony estimation and instrument recognition in polyphonic music. For many MIR tasks, it is common to use definitions taken from Music Information Retrieval eXchange (MIREX) [32]; however, there is currently no standard MIREX task definition for instrument recognition largely due to the ambiguity between producing frame-level or track-level predictions. We use the definition of instrument recognition by Li et al. [40], but modified to include polyphony estimation as well.

3.1 Polyphony Estimation

The definition of the term *polyphony* varies according to the MIR task of interest; in the case of chord or note recognition, polyphony refers to the number of notes sounding simultaneously. In this thesis, we define polyphony as the number of distinct types of instruments sounding within an *audio analysis frame* of fixed length (i.e., a small segment of audio of length which is known to have musically salient characteristics, chosen by the MIR researcher). Within an audio analysis frame, even if instruments do not overlap (i.e., there are *rests* or periods of silence present for a single instrument), we still consider the polyphony level to be the number of instruments sounding within the frame.

Additionally, to avoid confusion with other MIR tasks such as chord recognition, in this work we consider polyphony to be the number of *different instruments* sounding, rather than sounding of multiple notes from the same instrument, such as a 6-string guitar. The word *polyphony* in the context of number of instruments may also be used interchangeably with *voices* or *sources*. We also

do not consider multiple instruments of the same type to contribute individually to the polyphony number; as an example, if in one recording a single guitar is playing two different notes, but in another recording two different guitars are playing one note each from the previous notes, we still consider the number of distinct instruments to be one (i.e., guitar).

The *polyphony estimate* is the predicted number of instruments (or voices) present in a segment of audio, subject to a maximum polyphony constraint. More formally, given a segment of audio and a maximum specified polyphony p, the output is a one-hot vector $y \in \{0,1\}^{p+1}$ where, for $i = \{0, 1, ..., p\}$:

$$y_i = \begin{cases} 1 & \text{if polyphony is } i; \\ 0 & \text{otherwise.} \end{cases}$$

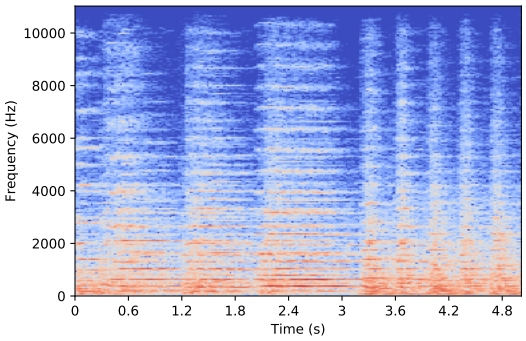
We use a one-hot encoding in order to facilitate classification. A limit on the maximum possible polyphony needs to be set when training a particular model for polyphony estimation by considering the polyphony distribution of available test data, and picking one maximum polyphony value that is representative of most tracks in the dataset (e.g., the mean polyphony across all tracks). In practice, the polyphony occurring in an audio analysis frame may vary drastically, from 0 (for periods of silence) to upwards of 20 (common in orchestral arrangements), both between tracks and within a track. Since a polyphony classifier needs to be trained on tracks up to a specified maximum polyphony, tracks with polyphony higher than this value will need to be discarded before training.

Polyphony estimation is a simpler problem than the more general task of fundamental frequency (f_0) estimation, since we are only interested in the number of simultaneously sounding voices, rather than estimation of the fundamental frequency values of each voice. One caveat is that predicting a single instrument from its multiple sounding may prove tricky.

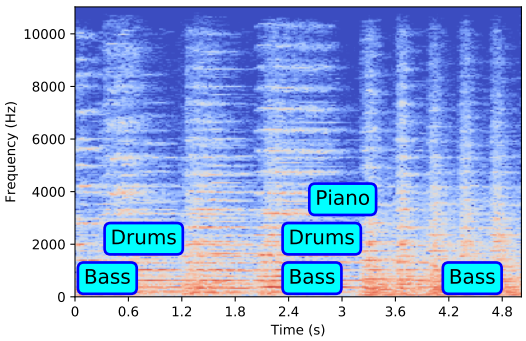
3.2 Polyphonic Instrument Recognition

Instrument recognition is an MIR task that can be performed in one of two ways: either recognizing the predominant instrument in a clip [3], recognizing all instruments playing in a clip, or recognizing multiple instruments sounding within a single audio window [49].

Instrument recognition may be performed in several contexts including images, video or audio . In this thesis, we limit instrument recognition to audio only, although video may provide visual cues helpful for polyphony estimation [69].



(a) Spectrogram of an audio excerpt.



(b) The same spectrogram, but with instrument onset occurrences labeled. Note the vertical positioning of the labels is only for clarity and does not have any additional meaning.

Figure 3.1: Spectrogram of audio excerpt and corresponding instrument labels.

Figure 3.1 shows a spectrogram of a music excerpt, and the spectrogram annotated with the occurrences of instruments at various time segments. Note the vertical positions of the instrument labels are only different for clarity and do not have any additional meaning in this context; only the horizontal coordinate is important.

We assume that the *instrument specification* (i.e., the types of instruments that may occur in an audio track for a given dataset of audio) is fixed. This ensures that we have a fixed number of output classes for instrument predictions. This, however, does mean that classifiers trained on data from a particular specification will likely perform poorly on test audio containing different types of instruments.

While instruments are predicted for each audio frame, we can choose to evaluate instrument recognition classifiers in one of two ways; either at the individual frame level, or at the entire track

level. In this thesis, we perform track-level aggregation, which identifies the presence of the most salient instruments occurring within the track, since evaluation may be performed using standard metrics defined in Chapter 2. It should be noted that the term *salience* refers to which types of instruments are noticeable in a given audio clip, regardless of the number of occurrences of each individual instrument.

Track-level prediction may be performed by either making a single prediction for the track as a whole, or making a prediction for each audio analysis frame within the track and then aggregating these frame-level predictions to form an overall track-level prediction. We choose to perform frame-level predictions in order to take advantage of signal processing techniques described in Chapter 2

The choice of aggregation method may affect the final prediction; two common methods are taking the average or the normalized sum of predictions over all windows [27]. Another possible scheme is a majority vote over analysis frames. We use the normalized sum technique which is described in more detail in Chapter 5.

The desired output given a frame of audio, a maximum polyphony p and a maximum number of instruments in the specified instrument specification l is given by a dense vector (in this case we do not use a one-hot encoding scheme since multiple instruments may sound in the same audio analysis frame) y where, for $i = \{0, 1, ..., l-1\}$

$$y_i = \begin{cases} 1 & \text{if instrument } i \text{ is playing anywhere in the frame} \\ 0 & \text{otherwise} \end{cases} \qquad 0 \leq i < l$$

subject to

$$p = \sum_{i=0}^{l-1} y_i$$

so that the polyphony constraint is met, otherwise the prediction is marked as incorrect.

3.3 Combined Problem Definition

We combine the problems of polyphony estimation and instrument recognition into a single problem as follows.

The input is an audio frame with a known maximum polyphony p and total number of possible instruments l according to a pre-defined instrument taxonomy. The output is a vector $y \in \{0,1\}^{l+p+1}$, a concatenation of polyphony estimation, and instrument estimation.

$$y_i = \begin{cases} \begin{cases} 1 & \text{if polyphony is } i \\ 0 & \text{otherwise} \end{cases} & 0 \leq i \leq p \\ \begin{cases} 1 & \text{if instrument } i-p-1 \text{ is playing} \\ 0 & \text{otherwise} \end{cases} & p < i \leq l+p+1 \end{cases}$$

An example output vector for a maximum polyphony of 3 and a total of 4 possible instruments is [0, 0, 0, 1, 1, 1, 1, 0], where the first four elements of the vector are the one-hot polyphony vector indicating that the polyphony level is 3, and the last four elements indicate that instrument numbers 0, 1 and 2 are currently playing.

This is formulated as a multi-label classification problem since multiple instruments may be playing according to the maximum polyphony specified. If the predicted polyphony number and the total predicted number of instruments do not match, only the polyphony prediction is marked correct if it matches the true polyphony. Likewise, if the number and types of instruments predicted are correct, but the predicted polyphony is incorrect, only the instrument prediction is marked as correct. Note that since we use the MetaLabeler approach described in Chapter 2, we can simply select the top instruments rather than check for polyphony and instrument mismatches.

3.4 Evaluation Metrics

The following common classification evaluation metrics will be used in this thesis.

3.4.1 Definitions

In the context of binary classification, a sample is described as *positive* if its true label matches the predicted label, and *negative* otherwise. The *true positive* rate is the ratio of positively predicted samples which also have a positive true label. The *true negative* rate is the ratio of negative predicted samples which also have a positive true label. The *false positive* rate is the ratio of positively predicted

samples which have a negative true label. The *false negative* rate is the ratio of negatively predicted samples which have a positive true label.

3.4.2 Precision, Recall and F-measure

Precision is the fraction of all positive predictions that are correct.

$$\text{Precision} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalsePositive}}$$

Recall is the fraction of all correct predictions that are predicted as positive.

$$\text{Recall} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalseNegative}}$$

F-measure is the harmonic average of precision and recall. This enables precision and recall to be weighted equally by penalizing false positives and false negatively by the same amount.

$$\text{Fmeasure} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Chapter 4

Related Work and Methods

In this chapter, we provide a summary of related work in polyphony estimation (Section 4.1) and instrument recognition (Section 4.2). We also describe some techniques for data augmentation in audio and touch briefly on the usage of synthetically generated training data in machine learning applied to audio 4.3. Finally, we explain our choice of methods used in this thesis 4.4.

4.1 Polyphony Estimation

Polyphony estimation in music is a challenging task due to overlap between sound sources, in this case instruments, in both time and frequency domains (i.e., the challenge stems from estimating the number of sources sounding in the same time window). In this section, we present some common related methods for polyphony estimation.

4.1.1 Fundamental Frequency Estimation

Most pitched instruments produce tones within a pitch range that have fundamental frequency (f_0) values based on their individual characteristics. Multiples of this fundamental frequency, called overtones, may then be observed in the audio signal of the instrument being played when converted to the frequency domain. A notable exception, however, are some types of percussion instruments, which do not possess a discernible pitch (e.g., snare drum). Some pitched instruments (e.g., gong) do not produce harmonic sound; however, instrument recognition and polyphony estimation is still possible in this case [43]. This is in fact a more complicated task than polyphony estimation, but can

be made into an equivalent task by using the fact that once each f_0 value is estimated, the polyphony may be estimated by simply counting the number of distinct f_0 estimates. An issue arises when two instruments have the same f_0 value, or if one has an f_0 value that is a multiple of that of the other, but timbre differences also may have an effect on the estimated frequency [81].

Klapuri et al. [34] performed multiple frequency estimation by summing harmonic amplitudes. This involves computation of the salience of a candidate f_0 frequency through a weighted sum of amplitudes of partial harmonics. Klapuri [35] also performed this task by computing a human auditory model in which fundamental frequencies were iteratively cancelled from the mixed signal. In both these works, polyphony estimation was evaluated alongside multiple fundamental frequency estimation by finding a stopping point for iterative cancellation.

Yeh et al. [85] designed another approach for polyphony estimation by building a set of f_0 candidates at the audio analysis frame level. Once the f_0 candidates are estimated, polyphony estimation is performed by considering partial overlaps to select the most plausible set of candidates using a score function.

4.2 Instrument Recognition

In this section, we discuss prior work in both monophonic and polyphonic instrument recognition.

4.2.1 Monophonic Instrument Recognition

Monophonic instrument involves identifying the instrument playing in a recording subject to the constraint that only one instrument is playing from a specified set of possible instruments. This may be formulated as a multi-class, single-label classification problem.

It should be noted that these methods do not apply to the case where multiple instruments of the same type may be sounding; other approaches based on *timbre analysis* (qualities of individual instruments) or *sound-source separation* (separating individual sources from a mixture of sounds) may be necessary.

Eronen et al. [18] use a variety of hand-crafted features such as mel-frequency cepstral coefficients (MFCCs) for training non-parametric classifiers such as k-nearest neighbors (KNN). However, monophonic instrument recognition methods are too simple for our use in this thesis since they do not extend to polyphonic instrument recognition which is a multi-class and multi-label classification

problem. Even if instruments do not overlap within an audio frame, monophonic classifiers do not apply since they are trained on audio that has only one instrument playing across an entire track; thus, they cannot provide multiple instrument predictions for an individual frame.

4.2.2 Polyphonic Instrument Recognition

Polyphonic instrument recognition involves identification of multiple instruments sounding among a set of possible instruments in a polyphonic recording. Note that the polyphony depends on the audio window size in question; if two instruments play sequentially in a given window, both instruments are considered present in the window [49]. Due to removal of the monophonic constraint, this is a more difficult task than monophonic instrument recognition. This is formulated as a multi-class, multi-label classification problem.

Bosch et al. [3] employed sound-source separation for polyphonic instrument recognition, which has also been applied in audio signal processing domains other than MIR. This involves treating the individual instruments in a polyphonic recording as separate sources of audio sharing a common recording device, and the task is then transformed into separating the individual audio sources, which represent instruments.

More recently, Lostanlen et al. [44] approached the polyphonic instrument problem by training deep convolutional neural networks with time-frequency kernels for detection of individual pitches corresponding to instruments. This approach used novel network architectures with pitch-inspired weight-sharing strategies. We instead use off-the-shelf convolutional neural networks since we are taking a synthesis approach to the problem and thus do not have enough information about the characteristics and quality of synthetic audio to use more sophisticated architectures.

Despite the popularity of convolutional networks for learning audio representations, a set of 92 hand-crafted features designed by Slizovskaia et al. [69] for training support vector machine (SVM) and gradient boosting classifiers have yielded state-of-art results in polyphonic instrument recognition.

Li et al. [41] apply deep convolutional networks for end-to-end learning from raw audio without feature extraction.

4.3 Audio Data Augmentation

Data augmentation [80] is a sampling algorithm method that attempts to learn posterior distributions via latent variables for mimicking a dataset distribution. In the context of training data, it is the transformation of existing data by adding audio effects, or generation of new data resembling the existing data distribution to make more training and test examples without changing the corresponding ground truth labels. Data augmentation can be used to help simulate noisy test data. Thus, data augmentation is often necessary for training a model that generalizes better to noisy test data, as evidenced by successes on the ImageNet dataset [37].

Schluter et al. [65] utilize data augmentation with time-invariant audio transformations (i.e., transformations that do not affect the timing of audio events) including pitch shifting to improve classification accuracy on singing voice detection.

However, other work focuses largely on time-variant audio transformations. Mauch et al. [47] and McFee et al. [49] designed software frameworks to approach data augmentation in music as application of a sequence of audio transformations which may change the timing and pitch of audio events and the corresponding output labels (e.g., dynamic range compression). Salamon et al. [62] also designed a software framework for data augmentation in soundscape event classification (i.e., detection of events in audio such as clapping or thunderstorms).

There is often no access to large amounts of real annotated music recordings; augmentation with synthetic examples, such as generated music, may help in such cases. Shotton et al. [66] applied synthetic data generation to pose recognition from single-depth images (i.e., only width and height dimensions). Barbosa et al. [1] also used synthetic data generation for person re-identification (i.e., discovering the identity of person given a description of the subject's appearance).

There are several different operations that can be applied to data augmentation in audio, and each of these operations may in turn have parameters controlling the magnitude of the operation to be performed. The choice of operations to use as well as the magnitude to be applied in order to yield a well-augmented dataset is often not known *a priori* and requires trial-and-error.

Cubuk et al. [13] treated this choice of operations and parameters as a search space and utilized *reinforcement learning* (i.e., continuous learning from streams of experience through trial-and-error) algorithms to find an optimal set of data augmentation actions with associated parameters. While

this approach shows promise, in order to focus on the use of synthetic training data, we maintain a fixed search space of audio transforms and choose randomly from this space.

4.3.1 Label Variance in Transformations

In object recognition, a common task in computer vision, data augmentation techniques such as reflection, rotation and shearing cause deformation of image samples without changing the assigned labels. For a different task in the same domain such as object localization, the underlying labels for the sample may actually change since the bounding box for an object of interest will likely change with such transformations [60].

According to McFee et al. [49], audio transformations may similarly be either label-invariant or label-variant at the frame level. Operations such as time-stretching or pitch-shifting will change labels for instrument recognition at the frame level. However, other operations such as magnitude scaling should not normally affect assigned instrument labels (although it may affect labels for instruments that solely produce noise).

4.3.2 Generative Models for Audio

The machine learning models discussed so far in this thesis are *discriminative* models, which attempt to learn the conditional class probability $P(Y|X = x)$ (where Y is a random variable denoting the output class and X denotes a random variable of features of which x is an instance). On the other hand, *generative* models differ from discriminative models in that the joint probability distribution $P(X, Y)$ is learned instead of the conditional distribution. Once this distribution is learned, the conditional probability distribution may be determined using Bayes' rule.

Generative adversarial networks (GANs) [24] are a class of machine learning algorithms used for generating examples of data similar to those given in a training dataset. The goal is to perform unsupervised learning of features in the original data and use those to construct new examples similar to, but not exactly the same, as the original ones.

At their core, GANs consist of two neural networks, namely the *generator* and the *discriminator* competing in a zero-sum game. The generator aims to fool the discriminator by generating new examples of data that the discriminator cannot distinguish as being separate from real ones, while

the discriminator attempts to mark the generated examples as fake, and relay this information to the generator so that the generation process can be fine-tuned.

It is worth noting that the above zero-sum game between two neural networks is a process not quite as simple as backpropagation in a single neural network. This often leads to divergence between generator and discriminator losses if the hyperparameters of each network are not tuned appropriately, since incorrect tuning may lead to either the generator or discriminator dominating over the other network.

Generative models have widely been used for generation of realistic audio, a prominent example being WaveNet by van den Oord et al. [79] which achieved state-of-the-art performance in text-to-speech synthesis. Speed of audio generation was further improved by Donahue et al. with usage of GANs [16].

In addition, Samuel [63] explored generation of realistic polyphonic multi-instrument audio has been explored using generative models in conjunction with large collections of MIDI files. However, the ground truth MIDI files are from real songs and are thus not synthetic compositions.

Unlike generative models, our approach does not rely on the existence of a training set of audio for use in generating similar examples. Instead, we seek to generate musical training examples without the use of any training data using simple composition methods.

4.4 Methods used in this thesis

We choose not to employ f_0 estimation methods in this thesis due to the added overhead of estimating each frequency, when the quantity we wish to estimate for our task is simply the *number* of such frequencies.

Instead of using SVMs, we use deep convolutional networks for the purposes of this thesis since they allow for more flexibility in learning representations for multiple output classes compared to SVMs where one support vector must be learned for each sample in the worst case [23].

We do not directly learn from raw audio, but choose to extract feature representations before providing input to the CNN to focus more on the synthetic generation aspect of our work.

The goal of our approach is to complement and extend the use of data augmentation by training deep convolutional networks on synthetically generated clips of multiple instruments, which may improve MIR task performance of supervised classifiers.

In this work, we choose to apply only label-invariant transformations such as noise addition in order to keep the space of possible transformations small.

Chapter 5

Synthesis of Musical Examples for Multiple Instrument Recognition

In this chapter, we propose a simple algorithm for synthetic audio generation used to train deep convolutional networks for polyphony estimation and instrument recognition in music. We explain each step of the algorithm, including composition generation, audio rendering, feature extraction, data augmentation, network training, and evaluation. We perform hyperparameter tuning using a simple grid search, and show that continuous generation and training in a loop performs better than training only once on a dataset the same size as the loop dataset size. Finally, we evaluate our algorithm on both a synthetically generated test set and a test set of real music excerpts[1].

This chapter is relevant to music informatics developers and researchers working with datasets with limited amounts of annotation. They may wish to adapt the synthetic training methodology to their own domains and tune the generation pipeline appropriately, or incorporate their own expert knowledge.

5.1 Introduction

Following from our discussion on the lack of annotated data and labels for MIR, we explore the use of synthetically generated audio recordings based on a predefined specification of instruments as

[1]My personal contributions included hyperparameter tuning, training of synthetic models, writing of training and evaluation code, evaluation on real test datasets and manuscript writing.

training data for instrument recognition in polyphonic music. We generate multi-track compositions of instruments following General MIDI (GM) standard specification of instruments and render these to audio using a soundfont synthesizer. We apply deep convolutional neural networks trained on these synthetic recordings to the MIR tasks of polyphony detection and instrument recognition.

The main contribution of this work is that it demonstrates the effectiveness of deep classifiers, trained on synthetic generated examples, on MIR tasks where they share no training data yet still perform. This implies that existing MIR tools that employ deep learning can augment their training sets and improve their robustness with synthesized examples.

We tackle the data availability problem in MIR by proposing a simple, continuous and end-to-end pipeline for generation of synthetic audio for use as training data for polyphony estimation and synthetic recognition. We begin with explaining a simple method for generation of synthetic audio compositions using the MIDI standard in Section 5.2.1 and explain how these compositions are rendered to audio using a soundfont synthesizer. Next, we describe the continuous nature of our pipeline by using our unlimited supply of training data to generate and train a deep convolutional neural network in alternating steps in Section 5.2.2 with a brief description of the network used in Section 5.2.5. Our data augmentation procedure is described in Section 5.2.3, and spectrogram design is covered in Section 5.2.4. Hyperparameter tuning is performed with grid search in Section 5.2.6. Empirical evaluation is performed in Section 5.4.

5.2 Methodology

In this section we describe a simple algorithm for generating audio for training a polyphony estimation and instrument recognition classifier employing practically unlimited synthesis of training examples as music represented as MIDI files.

5.2.1 Data Generation

To overcome the difficulty of finding large annotated datasets which cover a wide variety of musical styles, our system generates synthetic training data in the form of MIDI compositions of instrument note events sampled at random following a uniform distribution over the complete General MIDI specification (GM1) [50] of 128 instrument numbers. It is worth noting that not all preset numbers correspond to instruments (some are sound effects, such as gunshots) and thus some extraneous

classes (i.e., not relevant to instrument recognition) exist in the resulting classifier as well as in the training data. In addition, there is some bias introduced by sampling uniformly over the 128 instruments since many are guitars and keyboards; very few ethnic instruments exist.

The polyphony of each composition is chosen uniformly at random from the range [0, 3]. We choose this range since performance on polyphony tasks is found to decrease significantly as the maximum polyphony of the track under consideration increases beyond this range. Each clip is assigned a polyphony label in the range [0, 3] and between 0 and 3 instrument labels. The entire clip is considered as a single audio segment for the purposes of training a classifier. We use the Python library *mingus*[2] to create MIDI files.

In the generation process, we first choose a subset of available instruments to include in the clip. The generated multi-track file contains one track for each instrument chosen. Notes are then placed in each track until a full 2 second bar is filled (120 BPM). Notes are placed contiguously; no rests are added. Notes range in pitch from C2 (MIDI 24. 65.41 Hz) to C7 (MIDI 84, 2093 Hz) and in duration from 1/4 to 1/16 of the clip. The *mingus* library ensures the selected note is actually playable by the instrument; if not, a new note pitch is sampled. The note pitches were chosen to be close to the instruments with lowest and highest fundamental frequencies, the bass and piccolo respectively. From this range, note pitches corresponding to the playable range for each instrument are selected. It is worth remembering that even though there is only one note sounding at a time per track, since there is one track for each instrument, the polyphony for the entire clip remains constant.

Once the MIDI files have been generated, they are rendered to WAV using *TiMidity++*[3] using the freely-available *Fluid R3 GM* soundfont [4]. Some of the files may be slightly longer than 2 seconds because the instruments have significant sustain as shown in Figure 5.1; to remedy this, the WAV files are then clipped to exactly 2 seconds in length using *ffmpeg*[5]. If the audio is less than 2 seconds in playing time, then the clip is padded with silence using *ffmpeg*. Although the end of the clip may have no instruments playing, we still assign the entire clip a polyphony label equal to the number of instruments randomly selected. The length of each clip needs to be kept constant for training our neural network.

[2] https://bspaans.github.io/python-mingus/
[3] http://timidity.sourceforge.net/
[4] https://packages.debian.org/stretch/fluid-soundfont-gm
[5] https://ffmpeg.org

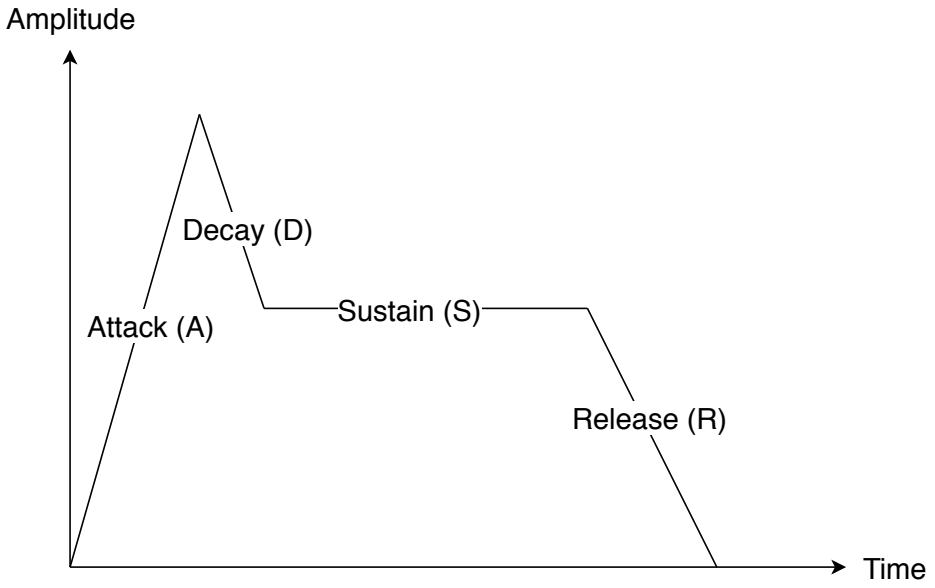


Figure 5.1: Sketch of typical note onset and offset profile, with the four stages labeled

5.2.2 Synthesis Loop

The synthesis loop focuses on generating new samples to train and test by generating music as described in the previous section, adding noise to the music, and then training and testing on the new music for some specified number of epochs, which we call *regen* for the remainder of this thesis. After *regen* epochs, the neural network is evaluated. The generated audio is then discarded, and the loop repeats. It should be noted that the purpose of discarding old clips and regenerating new clips after *regen* epochs is not to yield performance improvements on the test set, but rather to keep the process memory-efficient without having to save all clips on disk. The learning rate is exponentially scheduled to decay from a starting value of 0.1 as described below. Figure 5.2 shows a diagram of the operation of the synthesis loop.

5.2.3 Audio Transformations

Audio transformations are applied to the generated music with the use of csound[6], a program utilizing an audio domain-specific language (DSL). An effect from the following noise effects is chosen uniformly at random and applied to each sample.[7]

[6] http://csound.github.io/
[7] The noise program we used: https://github.com/abramhindle/audio-fuzzer

network: A polyphony/instrument classifier with learned weights
regen: Number of epochs between data generation
accuracy$\leftarrow 0$
tolerance$\leftarrow 10^{-4}$
learning-rate$\leftarrow 0.1$
while $\Delta accuracy > tolerance$ **do**
  generate labelled music
  generate noised versions of music, using the randomized noise functions of described in Section 5.2.3
  combine both datasets
  learning-rate$\leftarrow 0.1e^{-0.01*\text{epoch}} + 0.0000001$
  split into train and test sets
  train network for *regen* epochs
  accuracy$\leftarrow$ evaluate performance
end

Algorithm 2: Pseudocode describing operation of synthesis loop

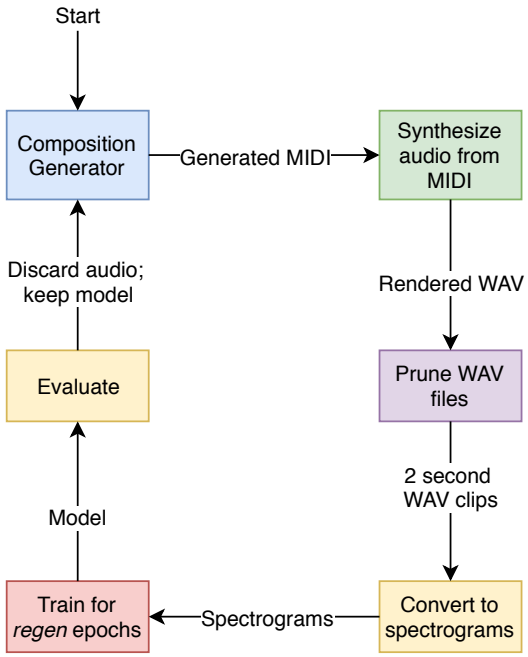


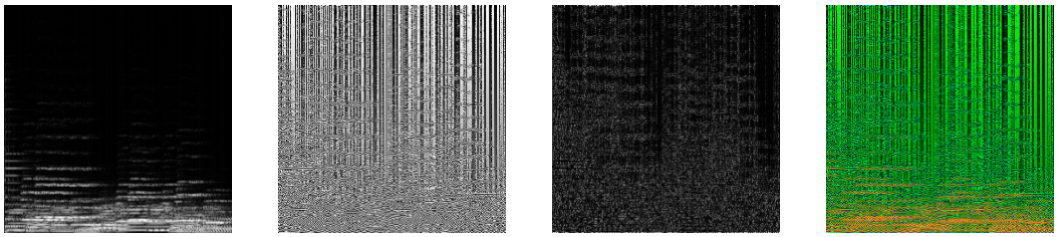
Figure 5.2: Diagram of synthesis loop

- reverb: random reverb added to the signal with a *roomsize* parameter (representing size of room) between 0 and 1, and a *damp* parameter (damping factor) between 0 and 1
- subtlereverb: a more subtle reverb added to the signal (same as reverb but with *roomsize* between 0.01 and 0.3, and *damp* between 0.9 and 0.99)
- tree: a small lowpass filter with cutoff frequency *cf* between 400 Hz and 4000 Hz

- manytrees: a lowpass filter meant to emulate trees dampening sound with a *cf* parameter (representing cutoff frequency) between 400 Hz and 1000 Hz
- high pass: a high pass filter at a random frequency between 60Hz and 1000Hz
- mid high pass: a high pass filter at a random frequency between 800 and 2000hz
- high high pass: a high pass filter at a random frequency above 2000hz
- whitenoise: white noise added to the signal, with *amp* (denoting amplitude) and *beta* (denoting a parameter controlling the window of the filter)
- hiss: lowpassed white noise added to the signal, with *amp* between 0.01 and 0.1, and *beta* between 0.5 and 1.0

5.2.4 Spectrogram Design and Input

The network input is a 227x227 (scaled down from 256x256 to fit network input dimensions without cropping, but keeping the same aspect ratio) spectrogram image generated using the short-time Fourier transform (STFT). 256 overlapping FFTs of size 512 are taken over the 2 second synthesized audio clip using a Hamming window, resulting in frequency bins of roughly 20 Hz in size. This results in multiple spectrograms per audio clip. We reduce the frequency range to between 100 Hz and 10000 Hz to reduce noise (note that we previously added audio transformations, but at lower cutoff frequencies than this range); although the cello has a minimum frequency of 65 Hz, most notes played are above this frequency. Log-amplitudes with linear frequency buckets are used for spectrogram creation. There are 3 channels of input: amplitude, phase, and gradient of phase with respect to time. Supplying the network with the phase gradient is intended to enable more precise frequency estimation, in a manner inspired by the phase vocoder. The 3 signals are then superposed onto each other as the RGB (red, green, blue) channels of a color image.



(a) Amplitude (b) Phase (c) Phase gradient (d) Overall spectrogram

Figure 5.3: RGB spectrogram composed of amplitude, phase and phase gradient channels

5.2.5 Network Architecture

We use the AlexNet [37] convolutional neural network architecture with weights pretrained on ImageNet [14]. The final softmax output layer is modified to have only $l+p+1$ output classes instead of the usual 1000 classes for ImageNet. A one-hot encoding is used for the polyphony class labels. The network output consists of two parts: first $n+1$ neurons, one-hot (encoded during training), indicating how many of the n instruments are sounding, then n neurons indicating the probability of each of the n instruments being present in the clip. The output is interpreted by predicting k instruments to be sounding, and then choosing the k instruments with highest probability.

For example given $l = 5$ (5 instruments) and maximum polyphony of 3, if the last 3 of the 5 instruments were sounding the vector y would be the concatenation of $[0, 0, 0, 1]$ (3 instruments) and $[0, 0, 1, 1, 1]$ (the last 3 instruments are playing) to form the vector $[0, 0, 0, 1, 0, 0, 1, 1, 1]$.

An output vector might look like $[0.1, 0.2, 0.3, 0.4, 0.0, 0.1, 0.9, 0.8, 0.7]$ which when processed with softmax on the polyphony would produce the expected output vector after normalizing.

For all training during experiments a stochastic gradient descent (SGD) optimizer is used with learning rate determined according to the schedule used in Algorithm 2.

5.2.6 Hyperparameter Grid Search

Hyperparameter and network selection is performed using a grid search over the following parameters, chosen since these ranges are commonly used when training deep neural networks (each repeated 5 times):

- Batch size: in range [5, 50, 500]
- Epochs: In range [10, 25, 50, 100, 200, 500, 1000, 2000]
- Loop: Whether we regenerate compositions continuously in a loop, or use a fixed set of compositions
- *regen*: In range [5, 10, 20, 50], the number of epochs after which data is regenerated if loop training is enabled
- Networks: One of [AlexNet, VGG-19[68], GoogleNet[70]]

Figure 5.4 shows a partial plot of the grid search described above.

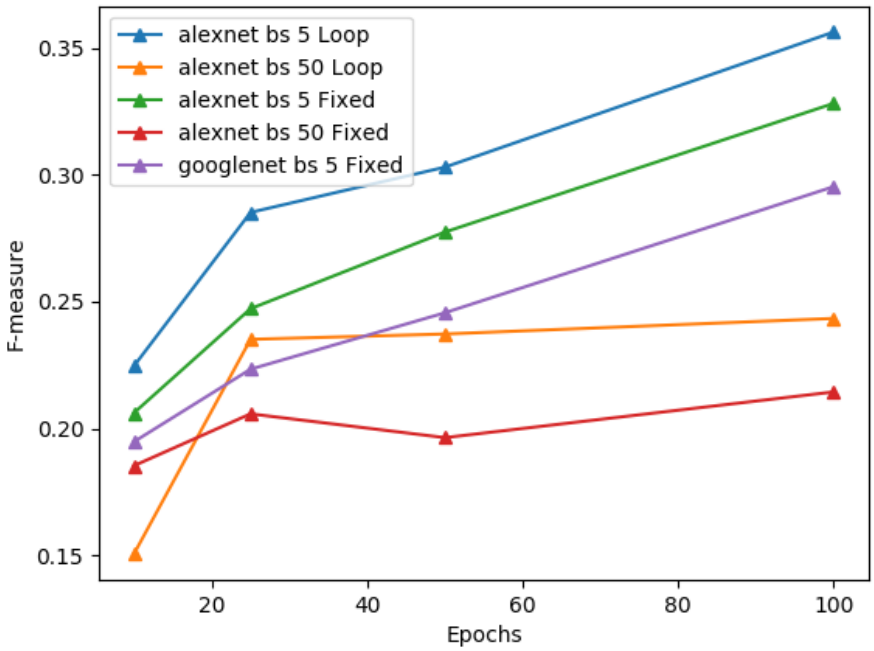


Figure 5.4: Plot of f-measure against number of epochs for different architectures, batch sizes and continuous versus fixed composition generation

Based on the best grid search performance, for all networks discussed in this paper we use parameters of batch size of 5, with a maximum of 100 epochs, using the AlexNet architecture with $regen = 10$. We also utilize early stopping with a loss tolerance (the minimum amount of change in the evaluation metric necessary for continuing training) of 10^{-4} and a patience of 3 epochs (the number of epochs after which training is stopped if no improvement in the evaluation metric is observed.

5.3 Evaluation

The evaluation metrics used on our polyphony/instrument classification experiments are (defined in Chapter 3:

- Precision
- Recall
- F-measure

Train-Test	Poly. Prec.	Poly. Recall	Poly. F-1	Inst. Prec.	Inst. Recall	Inst. F-1
Clean-Clean	$\mathbf{0.73 \pm 0.02}$	$\mathbf{0.73 \pm 0.02}$	$\mathbf{0.73 \pm 0.02}$	$\mathbf{0.35 \pm 0.02}$	$\mathbf{0.33 \pm 0.02}$	$\mathbf{0.34 \pm 0.02}$
Noisy-Noisy	0.53 ± 0.02	0.53 ± 0.02	0.53 ± 0.02	0.25 ± 0.02	0.21 ± 0.02	0.23 ± 0.02
Clean-Noisy	$\mathbf{0.65 \pm 0.03}$	$\mathbf{0.65 \pm 0.03}$	$\mathbf{0.65 \pm 0.03}$	$\mathbf{0.28 \pm 0.03}$	$\mathbf{0.24 \pm 0.03}$	$\mathbf{0.26 \pm 0.03}$
Noisy-Clean	0.52 ± 0.03	0.52 ± 0.03	0.52 ± 0.03	0.24 ± 0.03	0.21 ± 0.03	0.23 ± 0.03

Table 5.1: Test precision, recall and F-1 scores of clean and noisy models

These metrics are used instead of accuracy for both polyphony and instrument recognition in music evaluation due to class imbalances polyphony and number of instruments in the test dataset.

5.4 Results

5.4.1 Synthetic Test Performance

Our first task is to classify the polyphony and the instruments sounding of our generated music. Per each model, we perform 8 runs with 10,000 training samples (generated at random for each run), 2,500 validation samples (kept constant over the 8 runs for hyperparameter tuning), and 2,500 test samples generated once and held out (not used during training). We set the polyphony $p = 3$ and total instruments $l = 127$. The samples are generated in such a way that the numbers of classes of each polyphony are the same; there may be slight differences in the numbers of samples containing each instrument, but not enough to cause a statistically significant class imbalance problem since each instrument is equally likely to be chosen. This also ensures that a combination of instruments is very unlikely to occur more than once, but with the caveat that not all combinations of instruments may be present. The validation and test sets were kept constant across trials. For all experiments, we evaluate on the validation set at the end of each epoch to check for eventual overfitting within a tolerance level of 10^{-4}.

To investigate the robustness of synthetically trained networks to noise, we evaluate models obtained from spectrograms generated from clean WAV audio on test datasets with noisy audio and vice versa, where the noise added can be at random from any of the types described previously. The number of samples trained on is kept constant regardless of whether the model is trained purely on clean data or a mix of clean and noisy data (where the ratio of clean to noisy data is 1-1). The results are summarized in Table 5.1.

Table 5.2 summarizes polyphony F-1 scores obtained on a separately generated test set of 2500 MIDI samples for both clean and noisy models. For polyphony values of 1 and 2, clean models show

Train-Test	Polyphony	F-1
Clean-Clean	0	$\mathbf{0.99 \pm 0.06}$
	1	$\mathbf{0.80 \pm 0.06}$
	2	$\mathbf{0.60 \pm 0.05}$
	3	0.36 ± 0.08
Noisy-Noisy	0	$\mathbf{0.99 \pm 0.06}$
	1	$\mathbf{0.99 \pm 0.06}$
	2	$\mathbf{0.99 \pm 0.06}$
	3	$\mathbf{0.99 \pm 0.06}$
Clean-Noisy	0	0.96 ± 0.04
	1	0.15 ± 0.07
	2	0.01 ± 0.08
	3	0.85 ± 0.06
Noisy-Clean	0	0.99 ± 0.07
	1	0.15 ± 0.06
	2	0.01 ± 0.01
	3	$\mathbf{0.95 \pm 0.04}$

Table 5.2: Polyphony F-1 scores on test MIDI

poor performance when applied to noisy data and vice versa. This may be due to the additional noise effect being modeled as a sound source.

Despite the large number of output classes, both polyphony estimation and instrument recognition outperform a majority class baseline on a synthetically generated test set. As expected, models trained on clean audio perform best on clean test audio. Surprisingly, however, clean audio models applied to noisy data perform significantly better than noisy audio models applied to clean data.

The clean model did suffer much in terms of performance by being applied to noisy samples, while noisy data demonstrated more stability or robustness by maintaining performance on clean and noise evaluations although at generally lower performance.

5.4.2 Evaluation on real-world musical recordings

In order to demonstrate the feasibility of this method on actual recordings we engage in polyphony detection (the number of instruments sounding) and instrument recognition on existing benchmark datasets. We evaluate our synthesis loop algorithm on the IRMAS [3] dataset 2874 test samples in the form of excerpts from musical recordings ranging from 5 to 20 seconds in length containing a total of 11 instruments. We do not train on any of the IRMAS samples.

In order to make the classification on spectrograms resulting from clips longer than 2 seconds a well-defined task, we divide the clip into 2-second overlapping windows with 50 per cent hop size

and evaluate our synthesis loop model on each window. We calculate the mean frame-level F-score per instrument over all recordings. For IRMAS, the labels are predominant instruments, but since multiple instruments are labeled (note that not all instruments occurring in the clip may be labeled) per recording we may use the labels for multiple instrument recognition purposes. We performed a manual mapping from the instrument taxonomy of the dataset to the MIDI specification, which is described in more detail in Chapter 6. We count a prediction as correct if any of the MIDI instruments in the original instrument's mapping are predicted. The mean polyphony and instrument recognition scores along with standard deviation over 8 trials are found using the same 8 models trained on clean synthetic compositions as described in the previous section. The results are summarized in Tables 5.3 and 5.4.

Poly	M	Precision	Recall	F-1
1	C	$\mathbf{0.50 \pm 0.05}$	$\mathbf{0.56 \pm 0.04}$	$\mathbf{0.53 \pm 0.05}$
	N	0.24 ± 0.06	0.10 ± 0.06	0.14 ± 0.06
2	C	$\mathbf{0.37} \pm 0.06$	$\mathbf{0.24 \pm 0.04}$	$\mathbf{0.29 \pm 0.06}$
	N	0.10 ± 0.05	0.05 ± 0.06	0.06 ± 0.05
3	C	$\mathbf{0.75 \pm 0.03}$	0.14 ± 0.05	0.24 ± 0.06
	N	$\mathbf{0.75 \pm 0.07}$	$\mathbf{0.28 \pm 0.05}$	$\mathbf{0.41 \pm 0.07}$
Mean	C	$\mathbf{0.48 \pm 0.05}$	0.28 ± 0.06	$\mathbf{0.35 \pm 0.06}$
	N	0.41 ± 0.08	$\mathbf{0.29 \pm 0.09}$	0.34 ± 0.05

Table 5.3: Test polyphony precision, recall and F-1 scores on IRMAS. N for noisy models, C for clean models. Statistically significant results are bolded.

Polyphony estimation results from our synthetic classifier outperform a majority classifier, like ZeroR, at the frame level. The clarinet and trumpet classes yield the best average frame-level precision, while others such as flute fail to be positively classified. F-measure are not competitive with state-of-the-art approaches, Slizovskaia et al. [69] achieved 0.67 F-measure on IRMAS over all instruments. But the majority classifier is still outperformed for several instruments without any training on the IRMAS training set. Poor performance on some classes like voice may be explained by the lack of a standard MIDI program number for voice events. The noisy models did not perform as well on the IRMAS dataset. This may indicate that models trained on audio generated algorithmically from MIDI are not robust to audio transformations. While instrument recognition is not much better than a majority class rule, this may again be attributed to the large number of output classes the model was trained on.

Instrument	M	Precision	Recall	F-1
Cello	C	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	N	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Clarinet	C	$\mathbf{0.20 \pm 0.03}$	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.02 \pm 0.02}$
	N	0.02 ± 0.01	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.02 \pm 0.01}$
Flute	C	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	N	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Guitar (ac)	C	$\mathbf{0.04 \pm 0.01}$	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.03 \pm 0.01}$
	N	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Guitar (el)	C	$\mathbf{0.02 \pm 0.01}$	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.02 \pm 0.01}$
	N	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Organ	C	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	N	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Piano	C	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	N	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Saxophone	C	$\mathbf{0.02 \pm 0.01}$	$\mathbf{0.01 \pm 0.01}$	$\mathbf{0.02 \pm 0.01}$
	N	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Trumpet	C	$\mathbf{0.82 \pm 0.20}$	$\mathbf{0.07 \pm 0.04}$	$\mathbf{0.13 \pm 0.06}$
	N	0.05 ± 0.02	0.03 ± 0.01	0.04 ± 0.02
Violin	C	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	N	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Voice	C	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	N	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00

Table 5.4: Test instrument precision, recall and F-1 scores on IRMAS. N for noisy models, C for clean models. Statistically significant results are bolded.

5.5 Discussion

Our synthesis loop algorithm currently generates training examples at random. We could focus more on achieving high performance on a specific dataset (e.g., different genre) and tune the loop to generate data more similar to the test dataset under consideration.

Although we evaluated on a dataset of real music recordings, the effects of noise were not investigated. Augmenting these datasets with noise would enable us to further test robustness of our synthesis loop algorithm to noise. An additional path to explore is comparison of our noising framework with other open-source frameworks utilizing both label-invariant and label-variant transformations [49] to investigate which types of noise effects and other augmentations yield the best improvements in performance.

It is worth noting that soundfonts such as the one used to render are mostly restricted to Western instruments and are typically rendered in 12-tone equal temperament. A different approach may need to be taken for non-Western music collections, such as using a database of real recorded instrument samples to generate training data.

We restricted the use of transfer learning in this work to the use of pretrained model weights on ImageNet for facilitating speedup in network training; note we do not compare performance against randomly initialized weights. Transfer learning, the application of a model trained on one domain applied to a different domain, has been shown to improve performance between related MIR tasks such as genre classification and music similarity when faced with a limited number of training examples [26]. Transfer learning has also been applied to cross-domain prediction between music and speech [12], where it is shown that models trained on speech emotion generalize well to detection of music emotion and vice versa. One possible investigation is to fine-tune a synthetically trained model on real music to determine whether similar performance may be achieved in a sample-efficient manner.

5.5.1 Threats to validity

Construct validity is threatened by our assumption that uniform sampling of MIDI instruments, note events and intervals for composition generation may generalize to real music, which may not necessarily be the case.

Internal validity is threatened by our usage of 2-second clips for training; it may be that longer clips are need for instrument generalization.

External validity is threatened by our usage of the IRMAS dataset for testing, which is primarily composed of Western music excerpts. It remains to be seen how synthetic models perform on non-Western music collections, and possibly other Western music collection containing instruments for which there is no corresponding instrument number in the MIDI specification.

5.6 Conclusion

In this chapter, we presented a simple algorithm for generating synthetic musical compositions for use as training data for deep convolutional networks. We apply this pipeline to two tasks, polyphony estimation and instrument recognition. Hyperparameter tuning is performed on a validation set of synthetic audio, and final evaluation of this pipeline is performed on a test set of synthetic audio as well a test dataset of real music clips without leveraging the corresponding training set. We achieve performance which is better than a ZeroR classifier with statistical significance but falls short of state-of-the-art methods.

The key notable conclusions drawn from this chapter are:

- Generated compositions may be used to render audio for training data.
- Continuous generation of audio examples and training deep convolutional networks outperforms a single training run.
- Synthetically trained models yield weak but statistically significant performance improvements on real music compared to a ZeroR classifier.
- Simple data augmentation techniques like noise addition do not augment synthetic audio for performance improvements, but rather degrade performance.

Chapter 6

Synthesis of Training Data for Real-world Music

In this chapter, we note the lack of competitiveness of our approach in the previous chapter with state-of-the-art methods in polyphony estimation and instrument recognition and identify the need to reduce the range of possibilities of instrument and note combinations when generating compositions for our synthetic audio pipeline. A reduction in the types of compositions that can be generated is necessary to better emulate real music found in test datasets. We formulate the problem of composition generation as a search problem across several possible ranges of parameters. We present one method of reducing the range of possibilities of synthetic composition generation by considering the presence of instruments in the test dataset of interest. We evaluate this reduction on a test set of real music and achieve performance improvements in instrument recognition.

6.1 Reducing Composition Generation Possibilities

As stated in the previous chapter, performance of our deep neural networks trained with synthetic music on real test audio is not competitive with state-of-the-art approaches. There are several areas of improvement that may be explored, but perhaps the most obvious of these is the beginning of the synthesis loop, namely the composition generation stage.

It is difficult to classify unseen data when there is no prior knowledge regarding the distribution, characteristics or features of the dataset. In particular, modelling instruments playing in music is challenging. This is especially true in the case of this thesis, where the actual training set of real audio is not used. In machine learning, inclusion of prior information relevant to the data in feature representations for training classifiers is often necessary for improving model performance when applied to a particular domain.

In our case, the number of possible compositions is too large for a given set of randomly generated compositions to resemble real musical training data. In practice, only a small range of compositions of notes played by instruments is considered to be music. When generating compositions from MIDI, the following ranges of parameters were chosen from in the previous chapter.

- Maximum polyphony (an integer in range 0-3, 0 indicating a period of silence)
- Instrument preset (an integer in range 0-127)
- Note pitch (an integer in range 0-127, first reduced to 24-84, then adjusted to the range of pitches for each individual instrument)
- Note velocity (an integer in range 0-60)
- Note onset and offset times (duration ranging from a sixteenth note to a quarter note)

This space of parameters leads to a very large number of possible compositions, of which most are not likely to sound like real music. We propose methods to reduce the size and/or dimensionality of this space for more realistic composition generation.

One method is train a model which is more useful for instrument recognition on the test dataset under consideration, simply by considering which instruments are present in the test set. This enables us to reduce the number of possible output classes by reducing the range of instrument preset numbers from the full set of 128 to a smaller number. We can thus narrow the possibilities for composition generation by only selecting at random from this reduced set of instruments.

6.1.1 Mapping from MIDI to Test Audio

Since we are using MIDI instrument presets as a proxy for real instrument classes, we need to perform some mapping from the set of output class instruments present in the test set to MIDI instrument

`mapping`: A one-to-many mapping of true instruments to MIDI instrument numbers
p: Desired polyphony of composition
for $i = 0, \ldots, p-1$ **do**
  `trueinst` ← true instrument selected randomly from test dataset output classes
  $\texttt{midinums}_i$ ← random MIDI instrument number from `mapping`[`trueinst`]
end
make composition from `midinums`

Algorithm 3: Pseudocode describing instrument sampling for composition generation

presets. Table 6.1 provides a mapping from the IRMAS dataset instrument labels to instrument presets in the MIDI GM1 specification.

True Instrument	MIDI Instrument Presets
Cello	43, 44
Clarinet	72
Flute	74, 76
Guitar (acoustic)	25, 26, 32
Guitar (electric)	27, 28, 29, 30, 31
Organ	17, 18, 19, 20, 21
Piano	1, 2, 3, 4, 5, 6, 7, 8
Saxophone	65, 66, 67, 68
Trumpet	57, 60
Violin	41, 45, 46, 47, 49, 50
Voice	53, 54, 55, 86, 92

Table 6.1: Mapping from IRMAS true instrument output classes to MIDI instrument presets

With this mapping, we perform sampling of MIDI instrument numbers for composition synthesis by first sampling at random from the true instruments, and then sampling again at random from the instrument presets mapped to the sampled instrument. Algorithm 3 provides pseudocode for our sampling method.

6.2 Experiments

We repeat our experiment from Chapter 5 on the IRMAS dataset, except this time we restrict the instruments used for synthesis to the 11 instruments present in the test data only rather than using all 128 MIDI instrument presets. All other parameters for the synthesis pipeline remain unchanged, except that the number of classes in the output layer of the convolutional neural network need to be changed to the number of instruments present in the test dataset. Note that we do not add

noise to the generated audio samples for this experiment, since in Chapter 5, noise addition caused degradation in performance. Table 6.2 summarizes the results for polyphony recognition, while Table 6.3 summarizes the results for instrument recognition.

Polyphony	Precision	Recall	F-1	Change
1	0.50 ± 0.06	0.21 ± 0.04	0.30 ± 0.07	-0.23
2	0.47 ± 0.07	0.47 ± 0.05	0.47 ± 0.05	$+0.18$
3	0.15 ± 0.04	0.44 ± 0.04	0.22 ± 0.06	-0.19

Table 6.2: Test polyphony precision, recall and F-1 scores on IRMAS. + for performance improvement, - for decrease in performance.

Instrument	Precision	Recall	F-1	Change
Cello	0.09 ± 0.03	0.67 ± 0.03	0.17 ± 0.03	$+0.17$
Clarinet	0.01 ± 0.02	0.18 ± 0.02	0.01 ± 0.01	-0.01
Flute	0.12 ± 0.03	0.11 ± 0.02	0.12 ± 0.02	$+0.12$
Guitar (ac)	0.22 ± 0.12	0.02 ± 0.01	0.16 ± 0.02	$+0.13$
Guitar (el)	0.43 ± 0.07	0.03 ± 0.01	0.11 ± 0.02	$+0.08$
Organ	0.15 ± 0.01	0.04 ± 0.02	0.06 ± 0.01	$+0.06$
Piano	0.26 ± 0.02	0.36 ± 0.01	0.30 ± 0.03	$+0.30$
Saxophone	0.03 ± 0.04	0.01 ± 0.01	0.01 ± 0.01	-0.01
Trumpet	0.01 ± 0.01	0.01 ± 0.01	0.01 ± 0.01	-0.12
Violin	0.18 ± 0.02	0.67 ± 0.05	0.28 ± 0.03	$+0.28$
Voice	0.54 ± 0.02	0.60 ± 0.04	0.57 ± 0.05	$+0.57$

Table 6.3: Test instrument precision, recall and F-1 scores on IRMAS. + for performance improvement, - for decrease in performance.

6.3 Discussion

The reduction of possible MIDI instrument preset numbers used for composition generation improves instrument recognition performance on test real music with statistical significance for voice and stringed instruments. From the per-class instrument recognition results, it appears that stringed instruments (e.g., guitar, violin) become easier to distinguish, but there appears to be no significant improvement for brass or woodwind instruments (e.g., saxophone, trumpet), where performance is worse than ZeroR (which would result in 0.09 F-1 score). One explanation for the lack of F-1 score improvement in brass/woodwind instruments may be that their sustain period of note playing may, in real music, be significantly longer than the 2-second synthetic audio clips generated, resulting in the deep convolutional network being unable to classify these instruments correctly. However, this does not explain the fact that voice may also have a sustain period up to 30 seconds [46], yet showed

markedly improved performance; it may simply be that it is easier to learn 11 decision boundaries between instrument classes rather than 128 for the full MIDI specification.

Additional inclusion of realistic test music composition information and reductions in the composition space may be performed, of which we suggest a few. The first reduction may be performed by taking into consideration the distribution of instruments in the test set averaged over all tracks, giving a probability histogram against which to perform random selection of instruments. The sampling algorithm described in Algorithm 3 would remain unchanged, except that the probability distribution of selection would no longer be at random. This probability distribution may be a better representative of instrument occurrences in real music than our uniform distribution. It is important to note that when computing this distribution, additional instruments in the test set not present in the algorithmic generation specification are ignored or excluded.

Another approach may be to generate compositions based on random *chords* instead of random note pitches. Unlike our random note method of composition synthesis, composition of tracks with chords may sound closer to how instruments are played. This may in turn be a first step towards application of melodic rules, rather than sampling at random.

Clustering of MIDI output classes to form a different instrument class taxonomy [63] may also prove effective if each of the formed clusters represent output classes that are easier to distinguish between than either individual MIDI instrument numbers, or the groups formed by our mapping in Algorithm 3 . For example, it might be difficult to distinguish between a violin and a viola, but a superclass containing these two (and possibly other) instruments may be easier to predict.

6.4 Conclusion

In this chapter, we describe a need to reduce the space of possibilities when generating compositions in order to improve instrument recognition performance on real music. We propose one such method which takes into account the instruments actually occurring in the test set. We explore a reduction of the space of compositions by limiting our synthesis loop in the previous chapter to generating compositions containing only those instruments found in the actual test data, rather than being agnostic to the test set. This results in F-1 score performance improvements of up to 0.57 for voice and stringed instrument types.

Chapter 7

Conclusion

In this thesis we presented a method for overcoming limitations in the amount of training data available for MIR in the form of a simple synthesis loop algorithm for generating batches of synthetic compositions to be used in training convolutional neural networks. These networks were trained to estimate the number of instruments sounding, as well as which instruments are sounding.

We show that our method results in weak, yet statistically significant performance compared to a ZeroR classifier on the tasks of polyphony estimation and instrument recognition in music without usage of any training data from a target dataset for polyphonic instrument recognition. The benefit of this work is that synthesis allows learning of some features from polyphonic music, as demonstrated by test performance on real-world musical examples.

This implies that synthesis has potential in MIR, as it opens up the possibility for synthetically training classifiers to achieve state-of-the-art performance in a variety of MIR tasks with limited training data available. It is possible that a combination of audio data augmentation techniques and synthesis can aid future MIR tasks.

7.1 Future research directions

We will now suggest some extensions to our algorithm to improve learning efficiency and bring performance closer to existing state-of-the-art methods.

Our method of composition generation is rather simplistic; more realistic generation of synthetic recordings can be explored by taking into account musical information other than notes, such as chords, keys, meters, note durations and volume.

In this thesis, simple data augmentation techniques such as noise addition and signal filters did not improve performance on real music; rather, performance was degraded for all polyphony levels and instrument classes. It may be the case that synthetically generated music does not represent real music well enough to be augmented by audio effects; rather, whatever information synthetic clips contain may be obscured by the addition of noise. However, it would be useful to further study the effect of data augmentation on synthetic audio, in particular the effects of audio deformations other than noise and filters by comparing different open-source audio deformation frameworks. Synthetic data augmentation for musical score separation based on generation of multiple renditions of the score has been explored [72]; extension of this approach for varying renditions of a MIDI compositions may show promise.

The possibility of using transfer learning was not touched upon in this thesis. A route that can be explored is sample-efficient fine-tuning of trained deep audio classifiers. This would involve using the procedure described in this thesis as a pre-training step in MIR classification tasks, followed by further fine-tuning on real data. If performance comparable with state-of-the-art methods can be achieved with a fraction of the training data available, this means synthesis can be incorporated as a preliminary stage in several MIR workflows.

Another area that has not been investigated in this thesis is the effect of network architecture on performance. We modeled polyphony estimation and instrument recognition in tandem using a single deep convolutional network, meaning all intermediate layers and their weights were completely identical for both tasks. Alternative network architectures such as multi-input and multi-output networks could be explored where different initial layers are trained for polyphony estimation and instrument recognition (multi-input), but then combined in a way that preserves the polyphony constraint on instrument predictions (weight sharing). Multiple outputs corresponding to instrument predictions for multiple timescales could also be explored.

An interesting application of our methodology is in the field of ethnomusicology, in particular non-Western music databases. The soundfont (see Chapter 5 for details) used in this thesis for rendering synthesized compositions to audio only applied to Western instruments. Additionally, the test dataset of real music consisted primarily of excerpts from Western songs. It is worth investigating

whether our approach is also applicable to non-Western music, where the types of instruments and pitch scales used may vastly differ [77]; as discussed earlier, non-Western music does not normally use an equal tempered scale or place as much emphasis on music notation, which makes application of MIR techniques more difficult.

7.2 Summary

To conclude, this thesis presented a simple algorithm for generation of synthetic compositions as training data for MIR classifiers. We applied this approach to the tasks of polyphony estimation and instrument recognition. On a test dataset of real music excerpts, performance was weak yet better with statistical significance compared to a ZeroR classifier without using any real musical training data. An improvement to the algorithm was presented by using only test dataset instruments for composition generation, resulting in statistically significant improvements over ZeroR for voice and stringed instruments.

Bibliography

[1] Igor Barros Barbosa, Marco Cristani, Barbara Caputo, Aleksander Rognhaugen, and Theoharis Theoharis. Looking beyond appearances: Synthetic training data for deep cnns in re-identification. *arXiv preprint arXiv:1701.03153*, 2017.

[2] Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards AI. *Large-scale kernel machines*, 34(5):1–41, 2007.

[3] 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 *ISMIR*, pages 559–564, 2012.

[4] John Ashley Burgoyne, Ichiro Fujinaga, and J Stephen Downie. Music information retrieval. *A new companion to digital humanities*, pages 213–228, 2016.

[5] Gregory D Burlet. *Guitar tablature transcription using a deep belief network*. PhD thesis, University of Alberta, 2015.

[6] Michael A Casey, Remco Veltkamp, Masataka Goto, Marc Leman, Christophe Rhodes, and Malcolm Slaney. Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, 96(4):668–696, 2008.

[7] Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[8] Keunwoo Choi, György Fazekas, Kyunghyun Cho, and Mark Sandler. The effects of noisy labels on deep convolutional neural networks for music tagging. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(2):139–149, 2018.

[9] Keunwoo Choi, György Fazekas, Mark Sandler, and Kyunghyun Cho. Transfer learning for music classification and regression tasks. *arXiv preprint arXiv:1703.09179*, 2017.

[10] John M Chowning. Method of synthesizing a musical sound, April 19 1977. US Patent 4,018,121.

[11] Olmo Cornelis, Micheline Lesaffre, Dirk Moelants, and Marc Leman. Access to ethnic music: Advances and perspectives in content-based music information retrieval. *Signal Processing*, 90(4):1008–1031, 2010.

[12] Eduardo Coutinho, Jun Deng, and Bjorn Schuller. Transfer learning emotion manifestation across music and speech. In *Neural Networks (IJCNN), 2014 International Joint Conference on*, pages 3592–3598. IEEE, 2014.

[13] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

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

[16] Chris Donahue, Julian McAuley, and Miller Puckette. Synthesizing audio with generative adversarial networks. *arXiv preprint arXiv:1802.04208*, 2018.

[17] Valentin Emiya, Roland Badeau, and Bertrand David. Automatic transcription of piano music based on hmm tracking of jointly-estimated pitches. In *Signal Processing Conference, 2008 16th European*, pages 1–5. IEEE, 2008.

[18] Antti Eronen and Anssi Klapuri. Musical instrument recognition using cepstral coefficients and temporal features. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP'00. Proceedings. 2000 IEEE International Conference on*, volume 2, pages II753–II756. IEEE, 2000.

[19] Jonathan T Foote. Content-based retrieval of music and audio. In *Multimedia Storage and Archiving Systems II*, volume 3229, pages 138–148. International Society for Optics and Photonics, 1997.

[20] Emilia Gómez and Perfecto Herrera. Comparative analysis of music recordings from western and non-western traditions by automatic tonal feature extraction. 2008.

[21] Emilia Gómez, Perfecto Herrera, and Francisco Gómez-Martin. Computational ethnomusicology: perspectives and challenges, 2013.

[22] Michael Good et al. Musicxml: An internet-friendly format for sheet music. In *XML Conference and Expo*, pages 03–04, 2001.

[23] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[24] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[25] Masataka Goto et al. Development of the rwc music database. In *Proceedings of the 18th International Congress on Acoustics (ICA 2004)*, volume 1, pages 553–556, 2004.

[26] Philippe Hamel, Matthew E. P. Davies, Kazuyoshi Yoshii, and Masataka Goto. Transfer learning in mir: Sharing learned latent representations for music audio classification and similarity. In *14th International Conference on Music Information Retrieval (ISMIR '13)*, 2013.

[27] 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(1):208–221, 2017.

[28] Fredric J Harris. On the use of windows for harmonic analysis with the discrete Fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[29] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[30] G. E. Hinton. Learning multiple layers of representation. *Trends in Cognitive Sciences*, 11(10):428–434, 2007.

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

[32] Xiao Hu, Jin Ha Lee, David Bainbridge, Kahyun Choi, Peter Organisciak, and J Stephen Downie. The mirex grand challenge: A framework of holistic user-experience evaluation in music information retrieval. *Journal of the Association for Information Science and Technology*, 68(1):97–112, 2017.

[33] Eric J Humphrey and Juan P Bello. From music audio to chord tablature: Teaching deep convolutional networks to play guitar. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6974–6978. IEEE, 2014.

[34] Anssi Klapuri. Multiple fundamental frequency estimation by summing harmonic amplitudes. In *ISMIR*, pages 216–221, 2006.

[35] Anssi Klapuri. Multipitch analysis of polyphonic music and speech signals using an auditory model. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):255–266, 2008.

[36] Gunnar Kreitz and Fredrik Niemela. Spotify–large scale, low latency, p2p music-on-demand streaming. In *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*, pages 1–10. IEEE, 2010.

[37] 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.

[38] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[39] H. Lee, R. Grosse, R. Ranganath, and A. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the International Conference on Machine Learning*, pages 609–616, Montréal, QC, 2009.

[40] Peter Li, Jiyuan Qian, and Tian Wang. Automatic instrument recognition in polyphonic music using convolutional neural networks. Technical report, arXiv, November 2015. arXiv:1511.05520 [Cs].

[41] Peter Li, Jiyuan Qian, and Tian Wang. Automatic instrument recognition in polyphonic music using convolutional neural networks. *arXiv preprint arXiv:1511.05520*, 2015.

[42] Thomas Lidy, Carlos N Silla Jr, Olmo Cornelis, Fabien Gouyon, Andreas Rauber, Celso AA Kaestner, and Alessandro L Koerich. On the suitability of state-of-the-art music information retrieval methods for analyzing, categorizing and accessing non-western and ethnic music collections. *Signal Processing*, 90(4):1032–1048, 2010.

[43] Arie Livshin and Xavier Rodet. The significance of the non-harmonic "noise" versus the harmonic series for musical instrument recognition. In *International Symposium on Music Information Retrieval (ISMIR'06)*, pages 1–1, 2006.

[44] Vincent Lostanlen and Carmine-Emanuele Cella. Deep convolutional networks on the pitch spiral for musical instrument recognition. *arXiv preprint arXiv:1605.06644*, 2016.

[45] Michael I Mandel and Dan Ellis. Song-level features and support vector machines for music classification. In *ISMIR*, volume 2005, pages 594–599, 2005.

[46] Jonathan Maslan, Xiaoyan Leng, Catherine Rees, David Blalock, and Susan G Butler. Maximum phonation time in healthy older adults. *Journal of Voice*, 25(6):709–713, 2011.

[47] Matthias Mauch, Sebastian Ewert, et al. The audio degradation toolbox and its application to robustness evaluation. In *ISMIR*, 2013.

[48] Rudolf Mayer, Robert Neumayer, and Andreas Rauber. Rhyme and style features for musical genre classification by song lyrics. In *Ismir*, pages 337–342, 2008.

[49] Brian McFee, Eric J Humphrey, and Juan Pablo Bello. A software framework for musical data augmentation. In *ISMIR*, pages 248–254. Citeseer, 2015.

[50] midi.org. General midi (gm 1), mar 2016.

[51] Meinard Müller. *Information retrieval for music and motion*, volume 2. Springer, 2007.

[52] Meinard Müller. *Fundamentals of music processing: Audio, analysis, algorithms, applications*. Springer, 2015.

[53] Meinard Muller, Daniel PW Ellis, Anssi Klapuri, and Gaël Richard. Signal processing for music analysis. *IEEE Journal of Selected Topics in Signal Processing*, 5(6):1088–1110, 2011.

[54] Givewell Munyaradzi and Webster Zimidzi. Comparison of Western music and African music. *Creative Education*, 3(02):193, 2012.

[55] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[56] Jean-Jacques Nattiez. *Music and discourse: Toward a semiology of music*. Princeton University Press, 1990.

[57] A. Oord, S. Dieleman, and B. Schrauwen. Deep content-based music recommendation. *Advances in Neural Information Processing Systems*, (26):2643–2651, 2013.

[58] Herbert Robbins and Sutton Monro. A stochastic approximation method. In *Herbert Robbins Selected Papers*, pages 102–109. Springer, 1985.

[59] Howard Percy Robertson. The uncertainty principle. *Physical Review*, 34(1):163, 1929.

[60] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[61] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[62] Justin Salamon, Duncan MacConnell, Mark Cartwright, Peter Li, and Juan Pablo Bello. Scaper: A library for soundscape synthesis and augmentation. In *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2017 IEEE Workshop on*, pages 344–348. IEEE, 2017.

[63] David Samuel. Artificial composition of multi-instrumental polyphonic music. 2018.

[64] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. Current challenges and visions in music recommender systems research. *International Journal of Multimedia Information Retrieval*, 7(2):95–116, 2018.

[65] Jan Schlüter and Thomas Grill. Exploring data augmentation for improved singing voice detection with neural networks. In *ISMIR*, pages 121–126, 2015.

[66] Jamie Shotton, Toby Sharp, Alex Kipman, Andrew Fitzgibbon, Mark Finocchio, Andrew Blake, Mat Cook, and Richard Moore. Real-time human pose recognition in parts from single depth images. *Communications of the ACM*, 56(1):116–124, 2013.

[67] S. Sigtia, E. Benetos, N. Boulanger-Lewandowski, T. Weyde, A.S. d'Avila Garcez, and S. Dixon. A hybrid recurrent neural network for music transcription. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2061–2065, April 2015.

[68] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[69] Olga Slizovskaia, Emilia Gómez Gutiérrez, and Gloria Haro Ortega. Automatic musical instrument recognition in audiovisual recordings by combining image and audio classification strategies. In *Proceedings SMC 2016. 13th Sound and Music Computing Conference*. Zentrum für Mikrotonale Musik und Multimediale Komposition (ZM4), Hochschule für Musik und Theater Hamburg, 2016.

[70] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.

[71] Souhaib Ben Taieb and Rob J Hyndman. A gradient boosting approach to the Kaggle load forecasting competition. *International journal of forecasting*, 30(2):382–394, 2014.

[72] Naoya Takahashi, Michael Gygli, Beat Pfister, and Luc Van Gool. Deep convolutional neural networks and data augmentation for acoustic event detection. *arXiv preprint arXiv:1604.07160*, 2016.

[73] Lei Tang, Suju Rajan, and Vijay K Narayanan. Large scale multi-label classification via metalabeler. In *Proceedings of the 18th international conference on World wide web*, pages 211–220. ACM, 2009.

[74] Ole Tange et al. Gnu parallel-the command-line power tool. *The USENIX Magazine*, 36(1):42–47, 2011.

[75] Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Learning from between-class examples for deep sound recognition. *arXiv preprint arXiv:1711.10282*, 2017.

[76] Robert J Turetsky and Daniel PW Ellis. Ground-truth transcriptions of real music from force-aligned midi syntheses. 2003.

[77] George Tzanetakis. Computational ethnomusicology: a music information retrieval perspective. In *ICMC*, 2014.

[78] George Tzanetakis, Ajay Kapur, W Andrew Schloss, and Matthew Wright. Computational ethnomusicology.

[79] Aaron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[80] David A Van Dyk and Xiao-Li Meng. The art of data augmentation. *Journal of Computational and Graphical Statistics*, 10(1):1–50, 2001.

[81] Allan Vurma, Marju Raju, and Annika Kuuda. Does timbre affect pitch?: Estimations by musicians and non-musicians. *Psychology of Music*, 39(3):291–306, 2011.

[82] Avery Wang. The shazam music recognition service. *Communications of the ACM*, 49(8):44–48, 2006.

[83] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[84] Karthik Yadati, Martha Larson, Cynthia Liem, and Alan Hanjalic. Detecting socially significant music events using temporally noisy labels. *IEEE Transactions on Multimedia*, 2018.

[85] C. Yeh, A. Roebel, and X. Rodet. Multiple fundamental frequency estimation and polyphony inference of polyphonic music signals. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(6):1116–1126, 2010.

[86] Bo Yin, Eliathamby Ambikairajah, and Fang Chen. Combining cepstral and prosodic features in language identification. In *null*, pages 254–257. IEEE, 2006.