An Exploration into the Application of Convolutional Neural Networks for Instrument Classification on Monophonic Note Samples

James Owers

Master of Science by Research
Centre for Doctoral Training in Data Science
School of Informatics
University of Edinburgh

2016

Abstract

We show that instruments can be accurately recognised in monophonic, single note samples, using minimal temporal information. To classify each note sample, our method uses a convolutional neural network to predict the instrument for each frame of a Constant Q Transform, and simply takes the mean of these predictions. In contrast, state-of-the-art methods use hand engineered features to encode the temporal dependencies within a note.

By restricting the data it was trained on, we demonstrate that our model has the capability to generalise classification over the pitch range of instruments; it can classify the instrument for notes with a pitch that it has never before seen for that instrument. To our knowledge, we are the first to do this explicitly.

The high accuracy of our method (super human performance) suggests that there is enough information within short audio time frames to classify notes, and that CNNs can automatically learn features to leverage this information. Based on this observation, this work provides a solid basis from which to analyse the improvement made by temporal models with a similar structure, such as Recurrent Neural Networks. It also serves as the foundation for our future work, providing an architecture which we can extend to classify polyphonic, multi-instrument signals, or operate on lower level, raw waveform data.

Acknowledgements

I would like to thank my supervisor Amos Storkey for his generosity, sharing long and fruitful discussions, full of ideas. Thanks also to my coursemates for their support, in particular Jon, a sounding board, and a great critic.

Special thanks go to all the academics and professionals whom responded promptly, positively, and with much enthusiasm to my requests for information about their work, without exception. A further thanks to those anonymous people who contribute to open source software.

Finally, thanks go to: my friends whom have suffered and to whom I owe so much, my brother for always going one better, and my parents for their unwavering support and love.

This work was supported in part by the EPSRC Centre for Doctoral Training in Data Science, funded by the UK Engineering and Physical Sciences Research Council (grant EP/L016427/1) and the University of Edinburgh.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*James Owers*)

Table of Contents

Chapter 1

Introduction

> What an odd thing it is to see an entire species—billions of people—playing with, listening to, meaningless tonal patterns, occupied and preoccupied for much of their time by what they call "music."
>
> *Oliver Sacks, Musicophilia*

1.1 What can we learn from music?

Why do humans make and enjoy music? Could it be an evolutionary advantage to take pleasure in the games one can play with creating, and listening to sound? Was it a foregone conclusion that we would end up practising pattern recognition in this way?

The manner in which almost any given person can recognise their favourite song within a few seconds, or separate, and focus on a single voice from a choir of singers, is still a fascinating mystery. Audio separation and classification encompasses the classic 'Cocktail Party' problem which asks: how does a listener select a voice of interest from a backdrop of dozens, all speaking simultaneously?

Music is a structured medium. It has hierarchical temporal structure: individual pieces are composed of sections, which are a combination of bars, themselves formed of notes, each of which are played with a rhythm, stressing regular beats. Similarly there is hierarchical structure in pitch: the quality which we use to determine whether one note is higher than another; a defining difference between the piccolo and the double bass. These regularities in sound, combined with readily available data, provide the machine learning researcher with a superb playground, in which to train, and test the capabilities of their models.

Musical signals have differences and similarities to other data, such as speech signals, and natural language. Like speech, in their rawest form, the data are a simple stream of numbers, representing the strength with which particles in the air are vibrating some

measuring device; conversely, they are generally more tonal, and are more regular. Like natural language, they have an inherent, and importantly, non-local structure: removing a harmonic can change the perception of the tone, or the detection of a repeated chorus can change our belief about the genre; removing a 'not' can change the meaning of a sentence.

1.2 Why is instrument classification useful?

Instrument classification is a subset of audio classification in general: the challenge of labelling audio signals predefined characteristics. If there can be multiple labels applied simultaneously, this is often also referred to as audio tagging.

In order that a signal composed from multiple, potentially independent, sources can be separated, it is often useful to identify what those sources are. Additionally, if the task is to classify the genre of a piece of music, identifying the instruments within the piece can make this task much simpler. Transcriptions of polyphonic signals (a setting where multiple notes could be playing simultaneously) with multiple instruments requires the listener to identify and track the notes played by each instrument.

1.3 What do state of the art methods do?

On the whole, the most successful methods undergo a two stage process: feature extraction, and model fitting. Feature engineering has been inspired by the mathematical properties of sinusoids, and by biological observations of the human ear and the brain.

1.3.1 Monophonic, single instruments, single notes

Accuracy for monophonic, single instrument, single note signals is already at a super human level. The state of the art systems look to leverage musical structure by tracking harmonics, using timbre information provided by Mel-Frequency Cepstral Coefficients (MFCCs), and by detecting temporal patterns such as the strength of the note attack.

1.3.2 Polyphonic, multiple instruments, multiple notes

Polyphonic, multiple instrument, multiple note signals are a much more challenging medium, and are the focus of current research into Automatic Music Transcription, Source Separation, and Audio Tagging. Instrument recognition has not reached human levels of accuracy in this setting. Systems have much more of a challenge, given that they must also identify the *location* of notes within the signal. Some state of the art approaches involve combining source separation, onset detection, pitch detection, and instrument classification models.

1.4 An alternative approach: learning features

Neural network models have the potential to be able to learn features from raw signals. They achieve this by having a deep architecture and, at each level, learning to identify higher, and higher level concepts. An example from object recognition is that a network is constructed such that the first level learns to find simple edges. The next level might learn parts of objects by combining those edges, such as a wheel. Another level up could combine those detected objects to, for instance, detect a car from the observation of wheels, and a bonnet.

1.4.1 What are the benefits of learning features?

Why would we want to do this? Why would we want to relinquish the control of defining the features that our algorithm should use? One reason is that a model which learns its own features is more transferable. By this, we mean that, since the algorithm itself learned how to identify low-level patterns directly from the raw signal, it may also be able to learn, and critically, re-optimise features for a different task, such as to classify speech. Hand engineered features are often designed with a particular use case in mind, and it is down to the classifier on top to adapt to the potentially sub-optimal design.

Another reason is to improve performance. When using hand-engineered features, progress is often limited by the creativity of feature designers. When features are woven into the modelling process this is not necessarily the case.

Finally, these methods perform better at scale. Extracting features is costly both in terms of space and time. If we want our system to be able to return results on the fly, we need for it to operate on the medium the data are originally stored in.

1.5 Project motivations and proposition

To proficiently identify instruments in recordings of any given length, in different recording environments, played by any given musician on an instrument by any given manufacturer, and potentially amongst other instruments, a model may well require contextual information. This context could take the form of, for instance, knowledge of genre, musical harmony, or even basic physics. But how much and, if any, what context is needed? To begin to address this question, we start from the lowest level.

We intend to construct a baseline from which to build. Our starting point is to consider the fundamental building blocks of music: individual notes. Each note played by an instrument is the combination of all the vibrations produced by it. Each distinct instrument vibrates in a slightly different manner. For instance: the vibration of a trumpeters lips, combined with the air they push through, create a standing wave vibration within the instrument; the Spanish guitarist plucks a string, starting a standing wave along the string, which in turn vibrates the air within the body. These vibrations can combine

in a complex way; the standing wave along the guitar string oscillates at a given frequency, but the guitar itself is excited by this energy, and itself vibrates at a different rate. These interactions contribute to the characteristics of the sound.

In future experiments, in order that we can determine how important temporal information is, we transform the signal to the time-frequency domain, creating a matrix representation, and make a classification for each slice of time, which we call a frame. It has no knowledge of a given frame's relative position within the sound; it treats each frame independently of all others. In this way, it has no high level temporal understanding of sound. Once we have constructed such a representation, we will have a baseline with which we can compare our models that *do* have a higher understanding.

We propose to use convolutional neural networks to identify patterns within preprocessed sound, in the form of spectrograms. Spectrograms are a visual representation of sounds, aiming to describe which frequencies are present at a given time. Our aim is to draw from the success of these models within image recognition, and apply these techniques here. We will then move on to extending similar models for mixed signals.

1.6 What we show

We construct experiments to explore how much predictive power is contained within a) the magnitude of frequencies relative to the fundamental frequency of the note, and b) how much more accuracy we can achieve by allowing our models access to the full spectrum of frequencies. We find that the knowledge of the absolute position of magnitudes within the spectrum give a big uplift, and conclude that information about pitch is integral to the identification of the instrument.

Given this information, we design and train a CNN which is able to classify notes at a super human level of accuracy given only individual frames. After this, we explore the ways in which it fails. Finally, in the conclusion, we describe our next steps, which outline the extensions of our model into the polyphonic setting.

1.7 Thesis outline

In Chapter 2 we give some background, discussing the state-of-the-art, latest challenges, new approaches. We also provide links to standard datasets, and a list of useful definitions for the reader. Chapter 3 provides technical details for the features and methods used in the literature, and that we use in our experiments. Chapter 4 outlines the design and motivation for each of our experiments. Chapter 5 summarises the results of the experiments and provides an in depth analysis. Finally, in Chapter 6, we make our concluding remarks, next steps for this project, and discuss ideas for other avenues of research which are related.

Chapter 2

Background

2.1 Applications

Whilst our focus is upon Instrument Classification (known in the literature as Instrument Detection or Instrument Recognition), it is linked with many other areas of Music Informatics. For instance, it can be used in conjunction with Automatic Music Transcription (AMT) systems to enhance performance of both, as in [17]. Source separation, when applied to musical signals, is an extension of instrument classification, where the goal is not merely to classify instruments at given time-points, but also to extract the wave contribution of that instrument. Some Instrument Classification systems use source separation as part of their pipeline: Tjoa *et al*., in [47], use Non-negative Matrix Factorisation (NMF) to generate temporal atoms, which they use to create features.

Mel-frequency Cepstrum Coefficients (MFCCs) are the standard features to use for most applications since they are widely reported to encapsulate timbral properties [45]. More recently, there have been attempts to use lower level data and learn deeper models. We discuss these below in the Section 2.2.

2.1.1 Instrument Classification

Chétry's 2006 PhD thesis [9] provides a summary in Chapter 2 of how well humans perform instrument classification. They summarise studies that were mostly performed on subjects with a reasonable level of musical training. Figure 2.1 shows a comparison: generally, when there are 10 or more instruments, humans pick the correct instrument about 40-60% of the time. They note, with reference to [44] (a study with 27 different instruments conducted on conservatory students), that misclassification of Saxophone, Clarinet, and double reed instruments are most prevalent. Finally, it is claimed that performance is greatly reduced by trimming the ends or beginnings of notes, one study showing a reduction of 59% to 35%.

Much of the progress to date has been achieved with the help of feature engineering.

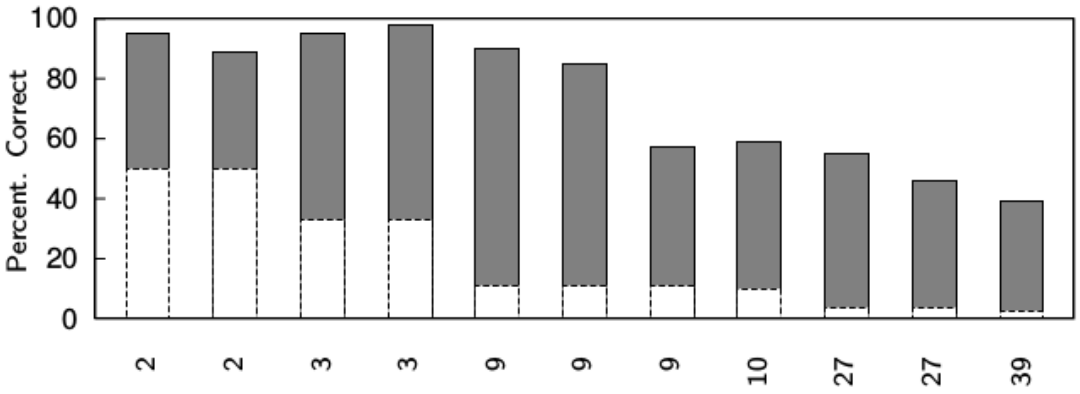


Figure 2.1: **Human instrument classification performance**: A comparison of different studies testing the ability of musicians to recognise instruments. The numbers on the x axis show the number of instruments used in the study. The white portion of the bar shows the accuracy that would be achieved by random guessing. Figure reproduced from [9].

Chapter 2 of Eronen's 2008 PhD thesis [14] provides both an introduction to features for instrument classification, and a comparison of approaches. We have reproduced, and added to this with some more recent results, in Table 2.1.

Arguably, Tjoa *et al*. produce the best results with their model described in [47]. They make use of MFCCs, combine these with temporal features, and use a Support Vector Machine to perform the classification. In [19], Grasis *et al*. take a different approach, aiming to produce a system that generalises to a polyphonic setting, using a combination of partial tracking, frame-wise, and note-wise features: they achieve similar results but test on fewer instrument classes. Partial tracking aims to detect continuous, thin frequency bands of activation within magnitude spectra, such as those produced by note harmonics.

Comparing these accuracies with human performance reported by Chétry, we can conclude that machine systems can outperform humans when classifying the instrument of monophonic single note samples. However, though Tjoa *et al*. and Grasis *et al*. train and test on multiple databases, they by no means explore a full range of recording qualities. Additionally, we draw attention to the fact that we have not found a standard dataset that the community has agreed to evaluate this task on. Hence, we hesitate to label this task as 'solved', though it has been widely studied.

Research into instrument classification for multi-note, polyphonic samples is ongoing. Giannoulis *et al*. put forward a 'missing feature' approach in [17] for polyphonic signals. This aims to break the problem down into multiple parts using a masking technique. When trained on the RWC database and tested on 10 Bach Chorales (4 parts played on separate instruments), the system achieves an average F-measure of 31% (bear in mind the system is also detecting the note locations). Abeßer & Weiß use NMF to separate sources and classify the instrument family of each in [1]. They achieve mixed results and do not detail their training and evaluation data.

Humphrey *et al*. take a different approach in [26]: starting with lower level features

from a Constant Q Transform (CQT), they fit a Convolutional Neural Network (CNN) optimised to create a three dimensional feature space (a non-linear semantic embedding) such that notes from the same instrument are close together. They do this by means of Siamese training: rewarding the network for placing two notes from the same instrument close together (and punishing if they are far), and vice versa. They use a simple k-nearest neighbour classifier on top of this, achieving accuracies of 98% on 5 classes. The perform a comparison with MFCCs and Principal Component Analysis (used for dimensionality reduction), which achieves just 63%. Of course, they could have improved performance with a more powerful classifier, but their aim was to show that a semantically meaningful embedding could be learned. We discuss this approach further in Section 2.2.

2.1.2 Automatic Music Transcription (AMT)

Automatic music transcriptions (also known as multiple fundamental frequency estimation and tracking, or simply multi-f0 estimation) can benefit from instrument classification techniques. Indeed, it is required if the input signal contains multiple instruments. Polyphonic, multi-instrument transcription is an area of active research. Benetos *et al*. give a review of the state of the art in [5]. They put forward a Latent Variable Model in [4] (which uses the CQT as input), and propose the combination of this with an explicit Instrument Classification model in [17], improving upon their transcription results.

NMF is often used to try and separate the spectrogram of a signal into individual components prior to transcription. Sigtia & Benetos explore the a different approach in [43] for polyphonic single instrument signals. In a similar manner to methods in Natural Language Processing, where they combine a translation model for words and a language model to smooth sentences, create an 'acoustic model' for notes, and a 'language model' to smooth phrases. They pre-process the data using a CQT. For the acoustic model they compare 3 layer neural network, CNN, and Recurrent Neural Network (RNN) approaches, and for the language model they use an RNN-NADE hybrid. They found that CNN outperformed RNN for the acoustic model and, when considering f-measure by time frame, and also outperform the state-of-the-art.

Though RNNs were employed by Sigtia & Benetos to model temporal relationships, no evidence was provided that long distance relationships were being learned (i.e. it is not clear that the problem of vanishing gradients was overcome); the fact that using RNN language model only presents a small increase in performance, over simply using the CNN acoustic model with binary threshold, hints that they were not. A possible extension would be to experiment with Long-Short Term Memory networks (LSTMs) and other RNN improvements to see if performance would be improved.

(a) *Same* recording conditions

Author year	Accuracy	Number of instruments
Kaminskyj 1995	98	4
Jensen 1999	100	5
Kaminskyj 2000	82	19
Fujinaga 1998	50	23
Fraser & Fujinaga 1999	64	23
Fujinaga 2000	68	23
Martin & Kim 1998	72 (93)	14 (5 families)
Kostek 1999	97 – 81	4 – 20
Eronen & Klapuri 2000	80 (94)	30 (6 families)
Agostini et al. 2003	70 (81)	27 (6 families)
Kostek 2004	71	12
Chetry *et al*. 2005	95	11
Park & Cook 2005	71 (88)	12 (3 families)
Humphrey 2011	98 – 84	5 – 12

Author year	Accuracy	Number of instruments
Martin 1999	39(76)	27(8 families)
Eronen 2001	35(77)	29(6 families)
Eggink & Brown 2003	66(85)	5(2 families)
Eronen 2003	68	7
Livshin *et al*. 2003	60(81)	8-16(3-5 families)
Peeters 2003	64(85)	23(7 families)
Tjoa *et al*. 2010	92	24
Grasis *et al*. 2013	91	11

(b) *Different* recording conditions

Table 2.1: **Comparing instrument classification systems in the literature**: Table (a) shows instrument classifications systems comparing notes from a single database with similar recording conditions, whereas (b) shows systems tested across multiple recording conditions. *N.B. Direct comparison of these methods is problematic as there is no standard dataset.*. Table updated from [14] with more recent additions.

2.1.3 Source separation

Source separation is a logical extension of classification: considering a signal as an additive mixture of signals, not only do we want to know when instruments play within a signal, we may also want to extract their contribution to the mixture.

A basic method for doing this in an unsupervised manner has already been presented above: pre-process the signal with a Short Time Fourier Transform or CQT, then use NMF to separate components. In [24], Huang *et al*. present a supervised method employing Recurrent Neural Networks to model Discrete Fourier Transform (DFT) and logged MFCC (logmel) features and Short Time Fourier Transform spectrograms. They achieve a good improvement on an NMF baseline. Audio examples are available on their website[1].

2.2 Feature learning

In this thesis, we are concerned with starting at a lower level and *learning* features with deep models. Humphrey, Bello, & LeCun make the case for this approach, with application to music informatics, in [25]. They state that progress in the field has slowed, citing lack of improvement in MIREX competitions, and that this was due to hand-crafted feature design being sub-optimal. To reinforce this, they point to Hinton *et al*. and their improvements on MFCC based systems within the Speech Recognition community in [22]. They also point out that, since the DFT and some filters can be represented as matrix multiplications, neural networks can represent them. Whilst they sing the praises of deep learning, citing its success in image recognition, they recognise "...there are many assumptions inherent to image processing that start to break down when working with audio signals". One such example is that nearby pixels correlate in images; frequencies are dependent in other ways e.g. harmonic patterns. This paper followed on from [26], mentioned above, where Humphrey *et al*. learn a non-linear semantic embedding for instrument classification.

As a slightly different example, learning features at a higher level, Hamel, Bengio, and Eck provide a method for learning hierarchical temporal features using a CNN in [20]. They build upon a spectrogram of MFCCs to learn temporal features. They use overlapping convolutions of varying sizes in the time dimension to learn features that are present over short, and long time periods. They achieve state of the art performance for tor that time.

Dieleman (the author of the deep learning package Lasagne) recently published their PhD thesis [11], in which they explore the use of Deep Neural Networks for audio tagging "exploiting the hierarchical structure of music". Whilst the focus was mostly on learning features from metadata, in Chapter 5 they explore 'end-to-end learning', where they train CNNs on mel-spectrograms and raw audio waveforms (adding a strided convolutional layer). Figure 2.2 shows some of the filters learned by the CNN when applied to raw audio.

[1] https://sites.google.com/site/deeplearningsourceseparation/

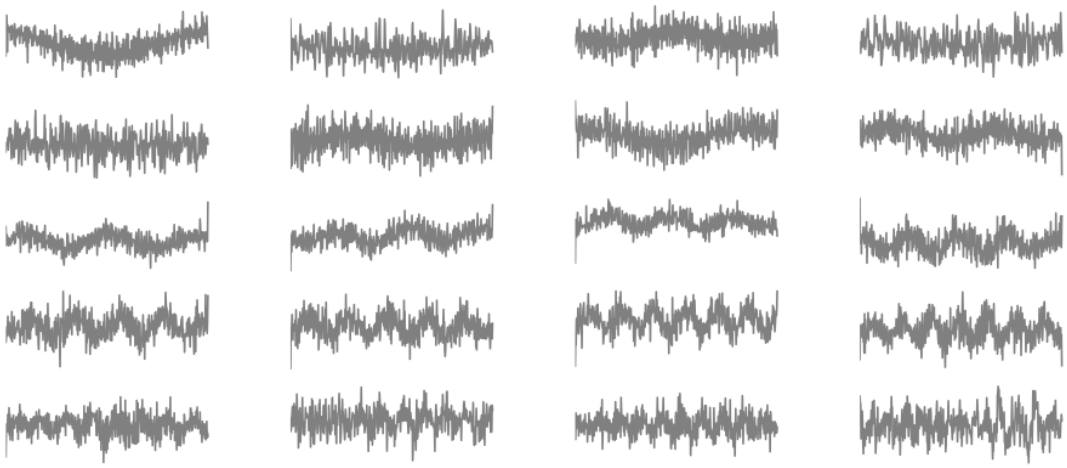


Figure 2.2: **CNNs applied to raw audio**: These are some of the filters learned by the first convolutional layer of a CNN applied to raw audio. Figure reproduced from [11].

2.3 Musical terminology with relation to signal processing

For the convenience of readers without a musical background we provide a list of useful definitions. For a description of methods see Chapter 3. Additionally, in Figure 2.3, we show a midinote – note name – frequency conversion table against a standard piano keyboard.

2.3.1 List of terms

harmonic When a single note is played by an instrument, this is in fact the sum of multiple frequencies. Typically, there are some stronger frequencies which relate to the perceived pitch, which we refer to as the harmonics. The strongest harmonics are located at integer multiples of the fundamental frequency.

f0 The fundamental frequency of a note: the typically the lowest frequency harmonic

octave The perceived interval between two notes when one has double the frequency of another.

semitone The base unit of pitch increase in western music. When a given note's f0 is multiplied by $2^{1/12}$, it increases by a semitone. there are 12 semitones per octave

note names There are 3 main ways in which we refer to the perceptual pitch of a note. We define them all with reference to concert pitch: a note with f0 = 440Hz. See Figure 2.3 for a visual comparison.

frequency The fundamental frequency of a note, f0

midinote An integer number representation, m. Concert pitch has $m = 69$. Generally, for a note with frequency f_m in Hz, $m = 12\log_2(f_m/(440\text{Hz})) + 69$

and for midinote m, $f_m = 2^{(m-69)/12} \times 440\text{Hz}$

scientific note name The note letter, followed by the octave number e.g. concert pitch is A4. Notes are lettered between A and G. Adding a '♭' lowers, and adding a '♯' raises the pitch by a semitone. Octaves are numbered starting from C i.e. A5 → B5 → C6. This leads to the strange nuance that C♭6 is an octave below B♯6

onset The beginning of the note: the sound from the instant the instrument is played, to the point of maximal amplitude [14]

partials Continuous activations within a magnitude spectrum, like those produced by a given note harmonic .

instrument range The set of notes that an instrument is able to play

instrument register The relative position of a note within the instrument range . High register indicates high pitch.

voice An individual instrument, or independent line within a instrument capably of polyphony, such as piano

phrase A collection of notes that are considered to be sounded by the same voice

monophonic musical compositions in which only one voice is present i.e. there are no overlapping phrases

polyphonic musical compositions in which multiple voices can be present

melody the aspect of musical composition concerned with the arrangement of single notes to form a satisfying sequence

harmony the aspect of musical composition concerned with the arrangement of multiple notes to form a satisfying sequence

timbre the character of a musical sound that is distinct from its pitch and intensity

2.4 Competitions

The Music Information Retrieval Evaluation eXchange (MIREX)[2] run yearly competitions for Music Informatics since 2005, and are normally held at the International Society for Music Information Retrieval Conference (ISMIR)[3]. The tasks this year included, amongst other things:

- Tagging 10s audio clips with labels indicating genres, instruments, and moods
- Genre, mood, and composer classification tasks on 30s audio clips
- Creating a musical similarity search engine algorithm which returns results based on an a piece of music, or singing/humming as the query

[2] http://www.music-ir.org/mirex/wiki/MIREX_HOME
[3] http://www.ismir.net/

- Multi-instrument, polyphonic f0 estimation (up to 5 instruments) on 30-sec audio clips
- Structural segmentation (e.g. A–B–A) for whole songs
- Singing voice separation from music backing for 30s clips

2.5 Standard datasets

The MIREX competition mentioned above is the main source for standard datasets. As discussed above before, we could not find a standard dataset for instrument classification. The literature we reviewed uses various different combinations and subsections of the following datasets:

MUMS McGill University Master Samples[4]
6546 WAV files of 3-10 seconds, divided between string (2204), keyboard (1595), woodwind (1197), percussion (1087, out of which 743 are nonpitched), and brass (463) families [13]

RWC The Real World Computing Music Database[5]
A musical instrument dataset of WAV files, covering 50 instruments, each played by 3 different musicians (playing instruments by different manufacturers) [18]. Additionally, 4 separate datasets including Popular, Classical, and Jazz music

MIS The University of Iowa Musical Instrument Samples (MIS)[6]
AIFF files of notes and scales covering the whole range of $>$ 30 orchestral instruments [15]

Good Universitat Pompeu Fabra good-sounds.org dataset[7]
Notes and scales from 12 different orchestral instruments created expressly to model differences in sound 'goodness' i.e. how well played an instrument is. Notes are played by multiple different musicians, each attempting to simulate good, and bad practice e.g. poor note stability [3]

OLPC One Laptop Per Child Free Sound Samples[8]
A very large and more eclectic selection of instrument and sound samples, including more diverse instruments such as didgeridoo, tablas, and beatboxing, as well as noises like footsteps and animal noises [35]

Freesound Freesound database[9]
An unsorted general sound database, containing some instrument sounds [31]

[4] https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mums/
[5] https://staff.aist.go.jp/m.goto/RWC-MDB/
[6] http://theremin.music.uiowa.edu/MIS.html
[7] http://mtg.upf.edu/download/datasets/good-sounds
[8] http://wiki.laptop.org/go/Free_sound_samples
[9] http://www.freesound.org/

There are also some commercial options, including Vienna Symphonic Library[10] (1.5 million samples), Logic Pro's Plug-ins and Sounds[11], and Kontakt 5[12].

For a more datasets used in the literature, see Appendix B.

[10] https://vsl.co.at/en

[11] http://www.apple.com/uk/logic-pro/plugins-and-sounds/

[12] https://www.native-instruments.com/en/products/komplete/samplers/kontakt-5/library/

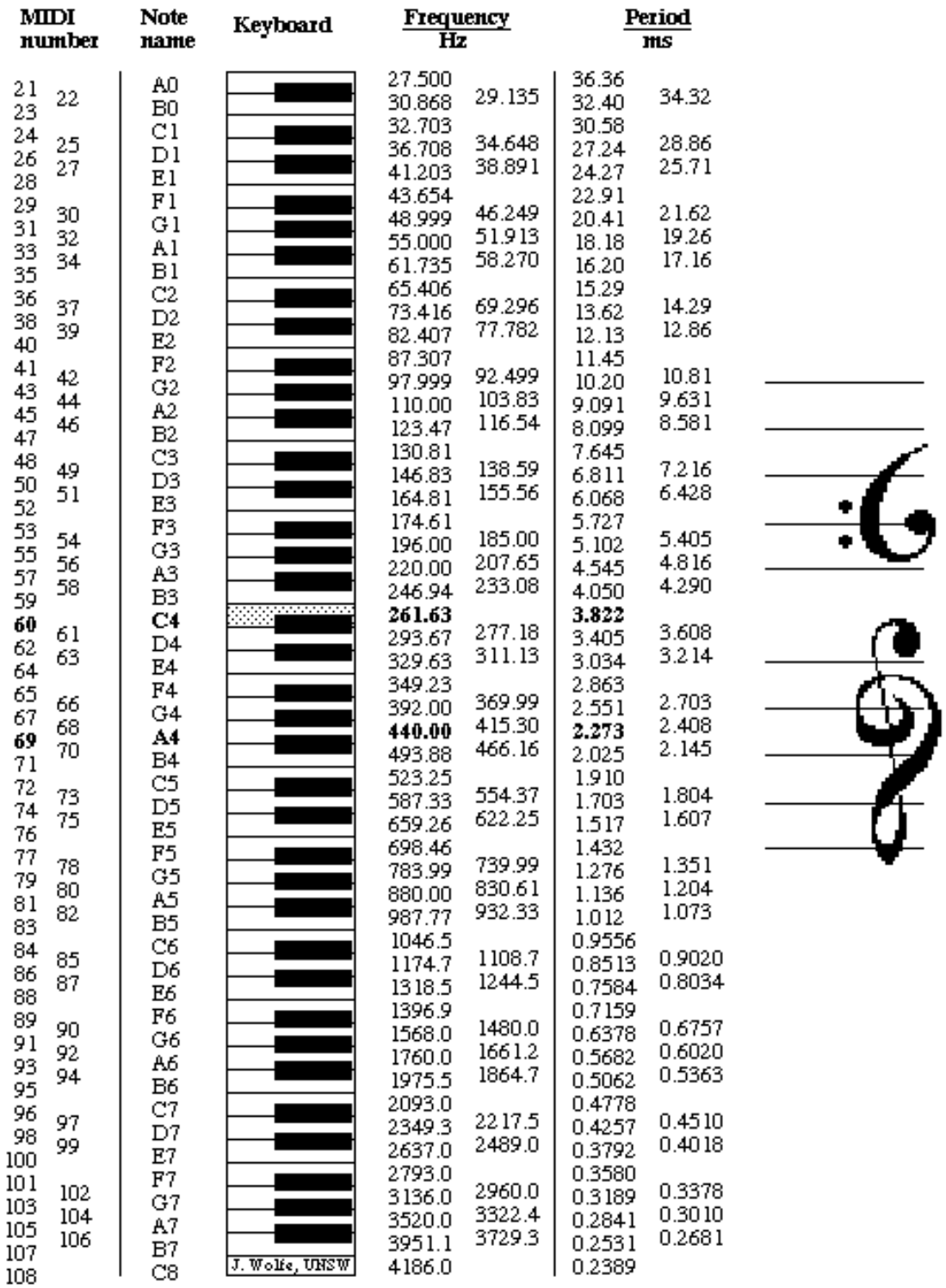


Figure 2.3: **Note notation**: reproduced from http://newt.phys.unsw.edu.au/jw/graphics/notes.GIF

Chapter 3

Methods

3.1 Data and features

3.1.1 Raw audio waves

We experience music as vibration, which can be represented as an amplitude wave. This is generally recorded by measuring the movements of a microphone diaphragm, but can also by synthesized. There are many raw audio file types. Some examples are WAV, AIFF, and MP3. They all amount to representing the wave as an array of numbers by defining the amplitude of the wave at discrete time points.

There are limitations to the accuracy of these data, determined by:

sample rate – the frequency at which samples are taken the e.g. 44.1 KHz

bit-depth – what numbers the amplitudes are allowed to be e.g. a bit-depth of 8 means that each number has only 8 binary bits of storage space, resulting in a vector containing numbers in the range [-128, 127] (for reference, audio is recorded at 24 bit in most studios)

channels – the number of streams recorded. As humans, we have 2 channel, or stereo hearing

These data are generally considered too low level for most models, and typically some pre-processing is done to the wave to create a higher level abstraction. One such example is the Discrete Fourier Transform

3.1.2 Discrete Fourier Transform (DFT)

The Discrete Fourier Transform relies on the Nyquist–Shannon Sampling Theorem, which, in the words of Shannon in [42], states:

"If a function x(t) contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced 1/(2B) seconds apart."

The DFT represents the samples as a set of sinusoidal frequencies. It assumes that the samples were taken from an infinite, continuous wave, and represents that continuous wave as the sum of sinusoids. Thus, any discrete sample of a wave can be completely determined by a discrete vector of frequency amplitudes (provided the conditions of the Nyquist–Shannon Theorem are met). A good, short explanation of this is given in this YouTube video[1].

For an $N \times 1$ dimensional vector of samples $\boldsymbol{x}$, the DFT vector $\boldsymbol{X}$ is an $N \times 1$ dimensional vector such that the k^{th} element is defined as:

$$\boldsymbol{X}^{(DFT)}[k] = \sum_{n=0}^{N-1} \boldsymbol{w}[n]\boldsymbol{x}[n] \exp\left\{-2\pi i k \frac{n}{N}\right\} \tag{3.1}$$

where $\boldsymbol{w}$ is some $N \times 1$ dimensional vector defined by the window function (e.g. Hanning), which weights the signal contribution according to the position within the window.

There is a problem with this representation for modelling: musical notes are actually mixtures of tones at multiples of the base frequency. This means that notes with a high base frequency get a sparse representation, notes with a low base frequency notes are very dense. See Figure 3.1 for a visual representation of this phenomenon. In contrast, the human ear appears to have approximately logarithmically spaced receptors for detecting vibrations.

3.1.2.1 Short Time Discrete Fourier Transform (STFT)

Normally, we want to produce a summary of a signal much longer than a few thousand samples; a typical pop song recorded at 44.1 KHz is a vector of the order of 10 million samples. Spectrograms are usually used for this purpose. They are the result of performing a transform on short, typically overlapping, sections of the wave. A typical setting for the STFT is to split the signal into sections of 1024 samples that overlap by 50% and perform a DFT to each. The overlap, is usually determined by the number of sample points between each window, refered to as the ***hop size***. A 'window function' is generally applied to each of the 1024 sample sections; this is to decrease the effect of leakage[2]. The benefit of this method is that there is an inverse transform (if the phase information is kept aside and reapplied).

3.1.3 Mel-Frequency Cepstrum Coefficients (MFCCs)

Mel-frequency Cepstrum Coefficients were originally developed in the 1970s and applied to speech recognition [30]. Their purpose is to address the problem of the non-linear nature of frequencies. Humans perceive pitch and loudness in a logarithmic way; the DFT gives equal precedent to subsequent frequencies, resulting in a representation more granular than humans detect as frequency increases and vice versa.

[1] https://www.youtube.com/watch?v=Cl-m4X3rwac
[2] http://www.robots.ox.ac.uk/~sjrob/Teaching/SP/l7.pdf

MFCCs are the most common features for pipelines in Music Informatics, with examples of use in modern systems for instrument detection [47], boundary detection [48], and source separation [24]. They are widely considered to represent timbre well [45].

The process to create MFCCs from a given audio signal is as follows:

1. perform a DFT on the signal and take the absolute values to obtain an $F \times 1$ dimensional vector of coefficients $\boldsymbol{f}^{(DFT)}$
2. choose N mel-frequency filterbank centers, and construct a set of N overlapping triangular windows, $\{\boldsymbol{w}_n\}_{n=1}^{N}$, or, considered as a $N \times F$ matrix $\boldsymbol{W}$ for convenience. The bandwidth of each window is the distance between its centre and the previous centre.
3. Transform the coefficients $\boldsymbol{f}^{(DFT)}$ to the Mel scale, simply by multiplying it by each window, i.e. performing the matrix multiplication $\boldsymbol{W}\boldsymbol{f}^{(DFT)}$, to obtain an $N \times 1$ dimensional vector $\boldsymbol{f}^{(MEL)}$
4. Perform a Discrete Cosine Transform on $\boldsymbol{f}^{(MEL)}$, decorrelating the coefficients and removing high frequency 'ripples' [45], to obtain the MFCCs

The first coefficients tend to describe the 'spectral shape' whereas the latter describe the pitch and more granular spectral details [14].

A comparison if different MFCC parameters is given in [51], and [45] gives a standard setting for application to music signals: use $N = 26$ Mel-frequency filterbank centres, and keep only coefficients 2-13 after the DCT.

3.1.4 Constant Q Transform (CQT)

The Constant Q Transform[3] was devised by Brown in 1991, and described in [7], with a further computational finesse in [8]. It attempts to create a logarithmic frequency representation of a signal with fewer transformations. This makes it appropriate for performing operations such as pitch shifting, as in [39].

The basic premise is that each frequency considered should be given the same priority by, at a given time point, considering a DFT window which could contain $\boldsymbol{Q}$ cycles of that frequency. In contrast, the STFT considers the same window size for all frequencies; each window could contain thousands of high frequency cycles (resulting in high, i.e. good, temporal resolution) and, in the worst cases, fewer than one low frequency cycle (resulting in low, i.e. poor, temporal resolution). In this way the STFT could be considered biased towards higher frequencies, and the CQT could be considered a fairer representation, more in keeping with human perception. In relation to western music, humans perceive a note to be an octave above another if it is double the frequency. Similar geometric patterns apply to other musical intervals, the most important being that a note increases by a semitone (the base unit of pitch increase) when it is multiplied by $2^{1/12}$ (there are 12 semitones per octave).

[3] It's not explicitly stated what the Q stands for, but Brown defines it in equation (2) of [7] and describes it as a 'quality factor'

For each frequency, f_k, considered, the CQT performs a function very similar to a DFT (shown in equation 3.4 below). The number of samples for the CQT is determined by 3 things: the frequency f_k, the audio signal's sample rate S, and the desired frequency resolution δf_k (i.e. which, in turn, defines Q_k).

The CQT insists that the ratio of each considered frequency to its resolution is constant:

$$\frac{f_k}{\delta f_k} = Q \quad \forall k \tag{3.2}$$

This is equivalent to saying that $f_{k+1} = (1+\delta)f_k \ \forall k$. In this way, once Q and a minimum frequency of interest, f_1, are chosen, the set $\{f_k\}_{k=1}^{K}$ are defined (the number of frequencies, K, is limited by the Nyquist frequency - half the sampling rate S)

For each frequency f_k, the number of samples being considered is defined as:

$$N(k) = \frac{S}{\delta f_k} = \frac{S}{f_k} Q \tag{3.3}$$

In addition to selecting a different set of frequencies to analyse, the CQT performs a slightly different operation to the DFT. For convenience, we repeat the DFT Equation 3.1, before stating the CQT equation for frequency f_k, highlighting the differences in red.

$$\boldsymbol{X}^{(DFT)}[k] = \sum_{n=0}^{N-1} \boldsymbol{w}[n]\boldsymbol{x}[n] \exp\left\{-2\pi i k \frac{n}{N}\right\}$$

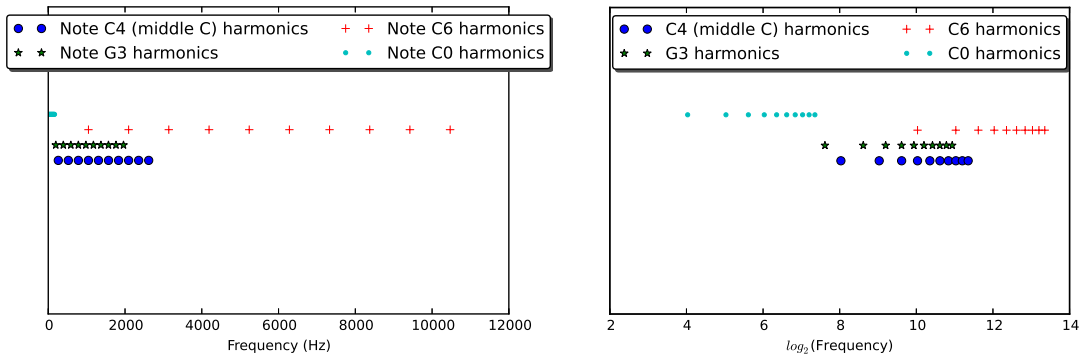
$$\boldsymbol{X}^{(CQT)}[k] = \frac{1}{N(k)} \sum_{n=0}^{N(k)-1} \boldsymbol{w}[n]\boldsymbol{x}[n] \exp\left\{-2\pi i Q \frac{n}{N(k)}\right\} \tag{3.4}$$

Brown notes that a resolution of $Q = 34$, corresponding to quarter-tone spacing, "...was still insufficient to resolve very high harmonics". Their solution was to double the Q value "for frequencies corresponding to G_6 (1568 Hz) and over".

To create a spectrogram, as in the case of the STFT, the experimenter can choose the locations time points within the signal at which to perform the transform. However, in the case of the CQT, one must be aware that the amount of overlap depends on the frequency. Where the STFT is like arranging overlapping rectangles, the CQT spectrogram is more like arranging overlapping triangles. This leads to an issue: successive lower frequency calculations could have a huge overlap, whilst successive high frequency calculations may not overlap, or worse, not consider some samples (if this is the case, the CQT is not invertible).

Whilst this is not an invertible transform, there are now more recent versions that are nearly [40], and perfectly [41] [49] invertible. The adaptations for perfect reconstruction suffer from impaired visual properties, namely that the spectrogram produced exhibits 'smearing' of the lower frequencies.

Some example CQT spectrograms are shown in Figure 4.1.



(a) (b)

Figure 3.1: **Notes and their harmonics**: The above plots show four separate notes and compare the frequency location of their harmonics. The distance between the harmonics depends on the fundamental frequency, denoted f0. In plot (a) we show the harmonics on a linear scale, and in plot (b) on a log scale. The harmonic distances are invariant to fundamental frequency on a log scale. This motivates the use of the CQT for convolutions.

3.2 Neural networks

Neural networks are flexible models designed to be a chained set of equations. Each successive level is a equation whose input is the output from some set of previous equations. Each equation can be seen as a node within the network. Typically, these nodes are organised into layers, such that each node within a layer is an equation concerning the output of nodes in the previous layer. When the output of all the nodes in one layer have been calculated, the calculations of all the nodes in the next layer can begin. At any given 'node' of the network, the main parameters of the model to be learned are the way in which its inputs should be weighted before summing. The result of this sum can then be transformed with some non-linear function if desired. An example is given in Equation 3.5, where the output value of the i^{th} node in layer 2 is $x^{(i)}_{\text{layer2}}$, the j^{th} node in previous layer, layer 1, is $x^{(j)}_{\text{layer1}}$, and the non-linear function is σ:

$$x^{(i)}_{\text{layer2}} = \sigma\left[\sum_{j=1}^{J} w^{(i)}_j x^{(j)}_{\text{layer1}}\right] \tag{3.5}$$

$$= \sigma\left[\boldsymbol{w}^{(i)} \cdot \boldsymbol{x}_{\text{layer1}}\right] \tag{3.6}$$

The equivalent matrix multiplication is also given, with $\cdot$ representing the dot product, and the vector $\boldsymbol{x}_{\text{layer1}}$ representing all the outputs of layer 1. The parameters to learn for node i in layer 2 are $\boldsymbol{w}^{(i)}$. We typically wrap these parameters up into the rows of a matrix $\boldsymbol{W}_{\text{layer2}}$ such that we can define the layer-wise calculation:

$$\boldsymbol{x}_{\text{layer2}} = \sigma\left[\boldsymbol{W}_{\text{layer2}}\boldsymbol{x}_{\text{layer1}}\right] \tag{3.7}$$

Layers such that each node within them are a use all the inputs from previous layer are called *fully connected*.

All that remains is to state that the input, $\boldsymbol{x}_{\text{layer0}}$, is known, and we have the means to generate output. By constructing a network which, in the last layer, has a single node, we can do regression. By adding a sigmoid non-linearity to that node, we can do binary classification. Similarly, an output layer with 10 nodes allows us to do 10 class classification, by means of a softmax non-linearity.

To learn the parameters for each node, the errors made when predicting output (defined by a loss function) are propagated back through the network to inform the direction of change for each parameter. Gradient descent is a robust, but slow method to achieve this. This is discussed in more detail below, in Section 3.3.2 on optimisation. One final thing for the reader to not is that, in the literature, the parameters are often referred to as 'weights', since they weight the incoming values in the linear sum.

3.2.1 How to learn features

Because of the flexibility to design complex architectures, the researcher can design a network to encourage it to learn different levels of abstraction from raw data. This could be considered similar to the process of hand-designing features. Where hand-designed features are tested by their ability to explain raw data, or improve the performance of models, the neural network is directly optimised to produce as efficient a representation it can for the task at hand. A good example of this is a particular architectural design called the convolutional layer, which we now discuss.

3.2.2 Convolutional Neural Networks (CNNs)

The *fully connected* layer we describe above is just one way of arranging nodes and the connections between them; the *convolutional* layer is another. CNNs, neural networks involving convolutional layers, were first formally described in [37] by Rumelhart, Hinton, and Williams. The motivation behind them is to detect patterns within data that has explicit local structure. For example, if we were aiming to classify pixels within an image black and white, we would like to encourage the network to look for local patterns. If the image is considered as a single, long vector, as it is by a normal fully connected layer, then there is prior on the features learned; adjacent pixels have no special relationship. Continuing the example, a convolutional layer restricts itself to considering 'patches' of the image. It reports the response of a bank of 'filters' (sometimes known as kernels), as they are passed across the image.

Imagine the image is a photograph negative, upon which is printed a 28×28 grid, with squares containing varying levels of opacity. We decide to use a convolutional layer with 8 filters to detect features. For the time being, we decide to design those filters ourself, though later, we will discuss how they are learned. Our filters are $3x3$ grids. We colour them black to represent 0 values, and leave them clear to represent 1 values (such that light can pass through). We place our photo negative on top of a light box,

and place one of the filters on top. The amount of light that passes through the filter is the response of that filter. We place the filter in every available 3×3 subgrid of the image, and record a value for the response, respecting the relative location from which it was taken. The result will be a 26×26 grid of values, which we call the *feature map*. If the filter was completely transparent (a matrix of 1s) then the feature map will report the sum brightness of each of the 3×3 squares - if we were to create another negative that looked like this feature map, it will look like a blurred, brighter version of the original image. What about if our filter was a 3×3 grid with a black, then transparent, then black column i.e. a column of 0s, a column of 1s, then a column of 0s? The feature map would be brightest where there were transparent vertical lines within the image. When we are done, we will have 8 feature maps of or image, each describing an aspect of the image based on the filter design, and retaining location information.

In reality, we deal with numbers in matrices. The operational analogue of 'detecting light' is simply to multiply the values in the filter by the image pixel values they cover, then sum the multiplications. Notice that, in the example above, each filter is just 9 parameter values to learn. Since we're using 8 of them, this is 72 parameters in total. We now have 8 feature maps though, which total $8 \times 26 \times 26 = 4608$ nodes. We, at the very least, need to connect them to an output node, which will require at least as many parameters. For comparison, a fully connected layer the with 8 nodes has $8 \times 784 = 6272$ parameters, but only results in 8 nodes.

3.2.2.1 Striding

We didn't have to slide the filter across every possible 3×3 grid either; we can use *striding*: the number of pixels to skip before the next observation. Striding with hyper-parameter (2, 2) means take every second observation horizontally and vertically. We may want to do this to reduce the number of nodes in the resulting feature maps, or to force the network to look for features in distinct, non-overlapping, places.

3.3 Neural network training techniques

Our attention now turns to learning these parameters. We first state some loss functions used for classification and regression, then discuss optimisation i.e. how to update the parameters such that we find a setting that minimises that loss function.

3.3.1 Loss functions

Loss functions, also known as cost functions, are what the network uses evaluate whether it is making its parameters better, or worse. The network as a whole can be considered as one big, differentiable function with a set of parameters Θ, but, for notational convenience, we will represent and refer to as a vector $\boldsymbol{\theta}$. The loss function is some function that, given a network architecture, depends on the parameters $\boldsymbol{\theta}$, the

input data used for prediction $\boldsymbol{X}$, and the labels we are trying to predict $\boldsymbol{y}$. We denote it $J(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{y})$, and are interested in how it changes as we change the values of $\boldsymbol{\theta}$ i.e. we want to minimise it.

N.B. we refer to elements of $\boldsymbol{\theta}$ *as* θ_k*, which we use we use to highlight that we are now talking about general architectures. In the context of the layered architectures described above, these are equivalent to the parameters* $w_j^{(i)}$.

A typical example loss function in the regression setting uses the mean squared error:

$$J^{(\mathrm{MSE})}(\boldsymbol{\theta};\boldsymbol{X},y) = \frac{1}{N}\sum_{n=1}^{N}\left(y^{(n)} - \mathrm{net}\left[\boldsymbol{\theta};\boldsymbol{x}^{(n)}\right]\right)^2 \tag{3.8}$$

Where there are a total of N data examples, and $\mathrm{net}\left[\boldsymbol{\theta};\boldsymbol{x}^{(n)}\right]$ is simply the output produced by the network with parameters $\boldsymbol{\theta}$ and single input data example $\boldsymbol{x}^{(n)}$. Put simply, the error is large if the network produces real number predictions that are different from the real number labels, and reduces as those differences reduce.

An example from the classification setting is called the cross-entropy loss and can be define as a vector quantity as follows:

$$\boldsymbol{J}^{(\mathrm{CE})}(\boldsymbol{\theta};\boldsymbol{X},\boldsymbol{y}) = -\sum_{n=1}^{N}\left(\boldsymbol{y}^{(n)} - \log\left\{\mathrm{net}\left[\boldsymbol{\theta};\boldsymbol{x}^{(n)}\right]\right\}\right) \tag{3.9}$$

This is a sum of vectors, where $\boldsymbol{y}^{(n)}$ is a 'one-hot' vector: a vector with dimensionality equal to the number of classes with 0 everywhere, except for the dimension representing the class of example n, which contains a 1. Each of the vectors in the sum is therefore zero everywhere except for the dimension of the true class label. If closer the probability for the true class predicted by the network ($\mathrm{net}\left[\boldsymbol{\theta};\boldsymbol{x}^{(n)}\right]$), the larger the contribution to the sum, which is then negated. Therefore the network is rewarded for predicting high probabilities for the true class.

3.3.2 Optimisation

Given that we have defined our loss function $J(\boldsymbol{\theta})$, we must decide how to update the parameters to minimise it. This relies on the entire network being a *differentiable* function.

3.3.2.1 Batch gradient descent

One simple and robust method is gradient descent. Put simply, we update each parameter in the direction that reduces the error, given the current gradient of $J(\boldsymbol{\theta})$ with respect to one of the parameters θ. But, given that we can calculate the gradient, how much should we update the parameter values? This is determined by a hyperparameter of the network, known as the *learning rate*, which we refer to as η.

For example, a given parameter $\boldsymbol{\theta}^{(i)}_{(layerOutput,j)}$ directly affects the value of the cost function (by affecting the output of the i^{th} final node by weighting the output of the j^{th} node in the layer below). We have the definition of the cost function and can differentiate it with respect to the parameter i.e. abbreviating the parameter as θ, we can calculate $\frac{\partial J(\boldsymbol{\theta})}{\partial \theta}$. We simply update $\theta^{(t)} \rightarrow \theta^{(t+1)}$ in the direction of the negative gradient:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \frac{\partial J(\boldsymbol{\theta})}{\partial \theta^{(t)}} \tag{3.10}$$

We have omitted many of the details about how to calculate the gradients, otherwise known as the backpropagation algorithm. The principle is the same for the lower layers: we always differentiate the cost by the parameter we are updating. This can involve many chained differentiations for deep networks with many layers, but the process amounts to being a special case of automatic differentiation. For further information about how the error is 'backpropagated' to update the parameters, we refer the reader to Chapter 2 of [32], which is available free online[4] and provides many further references.

We now discuss some alternatives to batch gradient descent.

3.3.2.2 Stochastic gradient descent

Batch gradient decent is based on a loss function that amalgamates errors over the *whole batch* i.e. all N training examples. This means that only 1 update is made to the parameters per epoch (iteration through every training example): this is pretty slow. Stochastic gradient descent makes an update after *every* training example it sees. The cost function is altered such that it concerns just one training example but, bar that, everything is the same. We now make N updates to the parameters every epoch.

This speed-up comes at a cost – inaccuracy. It's possible for the updates to the parameters to average in such a way that the descent follows a similar trajectory as for batch, but it is certainly not guaranteed. One classic 'gotcha' is that the order in which we present examples to the network suddenly becomes very important. What if we present all the examples with a specific class label first? The parameters will be updated towards an area of the loss space optimal for that class. This is particularly critical at the start of training, when errors, and therefore updates to parameters, will be large. Shuffling the data randomly is essential to ensure that the loss space is explored without bias.

3.3.2.3 Mini-batch gradient descent

If individual examples are too variant, even shuffling the data randomly may not provide a stable enough solution. Mini-batch gradient descent bridges the gap between

[4] http://neuralnetworksanddeeplearning.com/chap2.html

batch, and stochastic descent by breaking the batch into chunks and making a parameter update after each.

3.3.2.4 Issues with gradient descent

Gradient descent is guaranteed to descent to the global optimum if the loss surface is convex, and will converge to a local minimum otherwise. Aside from some special cases, we are normally dealing with a non-convex optimisation. There are other issues besides this:

- parameter updates get extremely small when the gradient gets near zero. This doesn't always happen near the minimum of the loss function, so gradient descent can get stuck in suboptimal places in the space (a classic example where gradient descent fails being the Rosenbrock function [36])
- each dimension of the data has a different variance, or in the case of categorical features a different frequency, so updating parameters with the same learning rate for all dimensions may not be optimal
- how the user is to choose the correct learning rate η is not defined other than to attempt to balance the speed and accuracy/eventual convergence of descent

3.3.2.5 Adagrad

Adagrad updates each parameter (at iteration t+1) $\theta_i^{(t+1)}$ at a different rate:

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{G}_{ii}^{(t)} + \varepsilon}} \cdot \frac{\partial J(\theta)}{\partial \theta_i^{(t)}} \tag{3.11}$$

Where ε is a small value to avoid division by 0, and $\boldsymbol{G}_{ii}^{(t)}$ is the sum of squared parameter values up to the current to epoch number t i.e.

$$\boldsymbol{G}_{ii}^{(t)} = \sum_{s=0}^{t} \left(\theta_i^{(s)}\right)^2 \tag{3.12}$$

We can use element-wise multiplication $\odot$ to define the update with respect to all the parameters:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{G}^{(t)} + \varepsilon}} \odot \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^{(t)}} \tag{3.13}$$

This updates the learning rate according to parameter changes which reduces the need to tune the learning rate. However, the value of each $\boldsymbol{G}_{ii}^{(t)}$ increases monotonically which results in an ever slowing rate.

3.3.2.6 Adadelta

Adadelta is used in [43] and was proposed by Zeiler in [50]. It is an extension of Adagrad which addresses the monotonically decreasing learning rate issue by simply allowing the optimiser to forget the parameter update history. It keeps a weighted average of the squared gradient updates:

$$\boldsymbol{g}_t = \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \tag{3.14}$$

$$\mathrm{E}\left[\boldsymbol{g}^2\right]_t = \lambda \mathrm{E}\left[\boldsymbol{g}^2\right]_{t-1} (1-\lambda)\boldsymbol{g}_t^2 \tag{3.15}$$

This is a weighted estimate of the root mean squared error of the gradient, updated at each epoch. It simply takes the place of $\boldsymbol{G}^{(t)}$ in Equation 3.13. Note that the square root term is simply an running estimate of the Root Mean Squared error (plus some small constant ε) so the shorthand $\mathrm{RMS}\left[\boldsymbol{g}\right]_t = \sqrt{\mathrm{E}\left[\boldsymbol{g}^2\right]_t + \varepsilon}$ is used to emphasize this. We define the change in a the parameters at time t to be:

$$\Delta\boldsymbol{\theta} = -\frac{\eta}{\mathrm{RMS}\left[\boldsymbol{g}\right]_t} \tag{3.16}$$

There is an added finesse which allows for the elimination of η entirely, thus relieving us of the duty of providing a learning rate all-together! Zeiler points out that the rate should have the same units as the parameter (this is not the case in any of the above methods), so makes an adjustment for by replacing η with an approximation for the Root Mean Squared difference in the *parameters*, to give the update equation:

$$\mathrm{E}\left[\Delta\boldsymbol{\theta}^2\right]_t = \lambda \mathrm{E}\left[\Delta\boldsymbol{\theta}^2\right]_{t-1} (1-\lambda)\Delta\boldsymbol{\theta}_t^2 \tag{3.17}$$

$$\mathrm{RMS}\left[\Delta\boldsymbol{\theta}\right]_t = \sqrt{\mathrm{E}\left[\Delta\boldsymbol{\theta}^2\right]_t + \varepsilon} \tag{3.18}$$

$$\Delta\boldsymbol{\theta}_t = -\frac{\mathrm{RMS}\left[\Delta\boldsymbol{\theta}\right]_{t-1}}{\mathrm{RMS}\left[\boldsymbol{g}\right]_t}\boldsymbol{g}_t \tag{3.19}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}_t \tag{3.20}$$

3.3.2.7 Adam

In [29], Kingma & Ba propose Adam, the optimisation we choose for our experiments. It builds upon Adadelta by adding a component similar to momentum, $\boldsymbol{m}_t$, helping the optimisation avoid local minima and saddle points.

$$\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t \tag{3.21}$$

$$\boldsymbol{v}_t = \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2 \tag{3.22}$$

where $\boldsymbol{v}_t$ is equivalent to $\mathrm{E}[\boldsymbol{g}]_t$ in Adadelta. Kingma & Ba add the additional step of a bias correction (away from zero) at the initial time-step, to give the update equation:

$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1-\beta_1^t} \tag{3.23}$$

$$\hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1-\beta_2^t} \tag{3.24}$$

$$\Delta\boldsymbol{\theta}_t = -\frac{\eta}{\sqrt{\boldsymbol{v}_t}+\varepsilon}\hat{\boldsymbol{m}}_t \tag{3.25}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \Delta\boldsymbol{\theta}_t \tag{3.26}$$

The authors recommend default settings of $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^8$ resulting for machine learning problems they tested, which we use.

For more information on optimisation methods for neural networks, we direct the reader to Ruder's excellent blog post[5] which provides comparison (including a useful animated visualisation), further methods, and more references.

3.3.3 Pooling layers

Pooling layers are a simple down-sampling operation after the calculation of the output values of the nodes in a layer. Like convolutions, they have a window size and a stride length. Two examples are max-pooling, and mean-pooling. For max pooling, the window is passed across the layer, with the given stride pattern, like a convolution, but just the maximum value is returned. This reduces the size of the layer and returns a 'sharper image'. Mean pooling is the same with the operation of a mean instead of max, and has the effect of reducing the layer size, and burring.

3.3.4 Dropout

Hinton *et al*. present dropout in [23]. It is a simple but effective idea to prevent overfitting which, in a similar way to ensemble methods, effectively averages the result of multiple, smaller networks. They achieve a big improvement on the state-of-the-art at the time. Hinton *et al*. suggested that the reason for the improvement is that it dissuades the network from learning "complex co-adaptations in which a feature detector is only helpful in the context of several other specific feature detectors".

The method is simple: upon presentation of a training case, randomly 'switch off' a random sample of neurons, dropping their links, and proceed as normal on the reduced size network. Dropout can be applied to any layer, including the input, or convolutions.

At test time, predictions can either be made using all the neurons (a rescaling of the parameters is required), or, as Gal & Ghahramani put forward in [16], sample the network parameters in exactly the same way as in training to make multiple predictions,

[5] http://sebastianruder.com/optimizing-gradient-descent/index.html

which can then be averaged. This second method has the added benefit of providing an uncertainty measure: the variance of the predictions. Gal & Ghahramani claim that this is akin to casting a neural network as a Bayesian approximation.

3.3.5 Batch normalisation

Neural network training can be sensitive to the initialisation of the parameters; starting in slightly different positions in the parameter space can result in converging to radically different solutions if the space is highly non-convex, as is often the case. Ioffe & Szegedy propose a solution to this in [27]. We provide a high level summary only here and direct the reader to the paper for further details.

By subtracting the mean of incoming data to the layer, and dividing by the standard deviation, the method normalises the incoming data, with the aim of keeping the distribution of the data the same regardless of the simultaneous other parameter updates. The network must learn the average mean, and average variance of the mini-batches during training, so that they can be applied at test time.

After our experiments were conducted, an extension, and potential replacement for batch normalisation, known as layer normalisation, was proposed by Ba, Kiros, & Hinton in [2]. We plan to update our method to incorporate this update in future.

3.4 Learning algorithms for comparison

To make inference about the data and contextualise the performance of the CNN we design in Section 4.5, we apply other models to our data. We briefly outline these below and direct the reader to the literature for further reading.

3.4.1 Linear and logistic regression

Since it was introduced above, we cast these models as simple neural networks. A linear regression is a neural network with an input layer the size of the input data, and one output node i.e.

$$y = \boldsymbol{w} \cdot \boldsymbol{x} \tag{3.27}$$

This has a convex loss function and a closed-form solution, so can be solved directly with matrix operations; there's no need for optimisation.

A logistic regression has exactly the same architecture, except for the output node has a sigmoid non-linearity function. This is no longer a convex optimisation, so optimisation must be used. See Equation 3.5 for the comparison with the neural network equations.

It's worth noting at this point we have not covered adding a bias term, but it is a simple matter of adding an additional dimension to the input $\boldsymbol{x}$ with the value 1.

3.4.2 Gaussian Naive Bayes

Naive Bayes is a generative model, which means that it models the input data space, conditional on the labels y. Each dimension of $\boldsymbol{x}$ is considered to be independent.

With all data from a given class i.e. $\{\boldsymbol{x}^{(n)} : y^{(n)} = k\}$ we fit a spherical Gaussian distribution via maximum likelihood, and a prior such that we obtain:

$$\mathbb{p}(\boldsymbol{x}|y = k) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k) \quad \forall k \tag{3.28}$$

$$\mathbb{P}(y = k) = \frac{|\{\boldsymbol{x}^{(n)} : y^{(n)} = k\}|}{N} \tag{3.29}$$

Given a new data example $\boldsymbol{x}$, and that there are K classes, we predict the label by employing Bayes' rule:

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\arg\max}\ \mathbb{P}(y = k)\mathbb{p}(\boldsymbol{x}|y = k) \tag{3.30}$$

3.4.3 K nearest neighbours (K-NN)

A K-NN model is a very simple model that retains the training data locations. At test time, it calculates the K closest training data points, and predicts either the most common label, for classification, or takes the mean label, for regression. As K increases, the model complexity decreases such that as $K \to \infty$ the classifier predicts only the mean value/class.

3.4.4 Decision trees & random forests

Decision trees create a partition of the input data space, and apply a classification, or real value to each. The basic idea for create the partition is to make successive, axis aligned splits by creating boolean operations based on the features. For example, if we have two dimensional data, we could split by the first dimension with an operation such as $x_1 \leq 10$. All data points $\boldsymbol{x}^{(n)}$ such that $\boldsymbol{x}_1^{(n)} \leq 10$ will be sent to the right leaf, and the others to the left leaf. At the leaves we make another split i.e. decision. Having fit a tree at training time, a classification is made by predicting the most common training label at each leaf. Similarly, in the regression setting, the mean value of training labels in the leaf is predicted.

The goal of training is to select splits that increase the 'purity' of the leaves: the ultimate split would perfectly put all of class 1 in one leaf, and all of class 2 on the right. We do not treat how they are trained in detail here, but direct the reader Breiman's text [6] for full details.

3.4.4.1 Ensembling trees

These simple models work very well when their predictions are combined in an ensemble. Two additions are made to the training process:

1. Each tree is trained on a bootstrap sample – a sample of size N with replacement, where N is the size of the training dataset; this technique is called bagging
2. The search for the best split at each node in the tree is randomized – instead of always considering all the features as candidates, only a random selection of features are chosen, and the optimal split chosen from them

In this manner, an ensemble, or forest, of variant trees is created. Whilst each tree can be seen as having a higher bias than one trained on all the data, and that can pick and feature to split on at each node, in fact we find dramatic improvement in performance. This is because complex signals can now be found that would otherwise have been dismissed by the optimisation because, for instance, one of the initial splits was 'sub-optimal'. If these complex signals are useful, and the forest is grown to a large enough size, these signals will be chosen by many trees, and amplified. If they are not useful, they will fade as the forest grows, or will be cancelled by opposing predictions from other trees.

This model design also has the benefit that each tree can be trained in parallel.

For a more in-depth discussion of random forests, see Crimini & Shotton's text on Decision Forests [10], in which they discuss many adaptations used in their work at Microsoft Research including for body part recognition using the Kinect.

Random forests (and their variants) are often amongst the best performing models in Kaggle competitions[6].

3.5 Evaluation Metrics

To compare our models, we use a few standard metrics for classification. We define them below.

3.5.1 Precision & Recall

Given a class of interest, recall is the measures what proportion of that class was correctly labelled. Precision measures the quality of the predictions of that class. One can easily obtain perfect recall for a class k by constructing a classifier that ignores the input and simply returns that class. However, in this case, precision will be poor, since the proportion of those predictions that were actually correct will be minimal. Similarly we can get perfect precision for k by designing a timid classifier that only predicts our class k when it is certain, perhaps only doing so for one or two examples

[6] https://www.import.io/post/how-to-win-a-kaggle-competition/

in the whole data set. The recall will be terrible, since we will have missed most of the examples of k that were shown.

$$\text{Recall for class k} = R^{(k)} = \frac{\text{\# correct classifications for class k}}{\text{\# class k examples in the data set}} \tag{3.31}$$

$$\text{Precision for class k} = P^{(k)} = \frac{\text{\# correct classifications for class k}}{\text{\# classifications made for class k}} \tag{3.32}$$

Creating a good classifier is about balancing these two measures.

3.5.2 F measure

The F measure, otherwise known as the F1 score, attempts to balance Precision and Recall, returning a single number. It is defined as their harmonic mean:

$$F_1^{(k)} = 2\frac{P^{(k)}R^{(k)}}{P^{(k)} + R^{(k)}} \tag{3.33}$$

3.5.3 Accuracy

For reporting accuracy, we disregard the class breakdown, and simply label each observation as correctly or incorrectly classified. The accuracy is the proportion of correctly classified examples.

Chapter 4

Experiments

We conduct these experiments to provide the grounding for our future work, modelling polyphonic, multi-instrument samples. We first identify the modelling challenges within the data by restricting the data space, then design a Convolutional Neural Network for instrument classification, which we label CNN3. The CNN3 will provide the proof of concept that features can be learned for instrument classification using nothing but individual frames of the CQT, treating them as independent data points. We start by modelling the fundamental building block of musical sounds: individual instrument notes.

Specifically, our experiments aim to determine whether there is enough information within short time periods of a note to classify the instrument playing it. We additionally investigate note regression though, since we find this to be a trivial task using a simple CNN (see Section 5.2.2), we move on from this quickly. We begin with a brief introduction to the data and preprocessing steps, then outline the three main investigations. The experiments aim to answer the following questions:

1. What is the predictive power of relative frequency?
2. How is performance improved by knowledge of absolute frequency?
3. Can a CNN learn features required to predict instruments?

The third experiment, in which we design and train CNN3, encompasses the body of the work in this thesis, with the first two allowing us to make observations about where complexities arise within the data.

4.1 Data

For all of our experiments we use the University of Iowa Music Instrument Samples (MIS) database [15]. This includes, amongst other things, recordings of the most common orchestral instruments playing every note within their range. Stringed instruments play all notes on each individual string in both arco (bowed) and pizzicato (plucked) playing styles. Some instruments, e.g. B♭ Trumpet, have examples of notes played

both straight and with vibrato. Each file contains one note, is approximately 10 seconds long, recorded at 44.1 kHz, and is saved as a Audio Interchange File Format (AIFF) file. The audio is of a high quality, having been recorded in an anechoic chamber and with a sensitive condenser microphone. Full details of the recording conditions are available on the website[1]. Appendix A provides details on obtaining the data.

4.1.1 Selecting instruments

To reduce processing times and the scope of our investigation, we discount some instruments and playing styles. In order that our classifier models the timbral properties of each instrument and not the playing style, we make the decision to exclude notes with embellishment e.g. vibrato. We state that, for stringed instruments, pizzicato and arco playing styles sound significantly different to be considered different instruments and cite that in [47] they are treated as such; we only include arco playing styles so as not to overly focus on stringed instruments. We exclude all percussion instruments (except piano) to restrict our investigation to tonal instruments only.

Our selection attempts to represent diversity within each family, both in terms of timbre and range, and includes 14 instruments in total: 1 Piano, 4 Brass, 4 Strings, and 5 Woodwind. The full list is contained in Table 4.1 below. Our intention is to revisit experiments with a greater range of instruments and playing styles at a later date.

We also restrict the f0 range of the notes that we consider in order that we attain a reasonable instrument overlap (i.e. there are at least 3 instruments for every tone in the dataset). The minimum note we select is C1 (midinote 4) and the maximum is E♭6 (midinote 67). With the Constant Q Transform (CQT) parameters stated below, this ensures that all notes have space for 11 harmonics.

A visual representation of the overlap in the range of the instruments is shown in Figure 4.3.

4.1.2 Preprocessing

For each AIFF file we create a spectrogram using a CQT, as described in Section 3.1.4, and randomly select 100 frames. Each frame summarises the frequencies occurring within a short time frame (between 1ms and 1s depending on the frequency) around a given point in time. Each resulting data example produced we refer to as a frame, and is a vector of size 216. We then label these frames with the note f0 pitch name and instrument name, and divide them into 3 parts for training, validation, and testing.

4.1.2.1 Application of the CQT

We use the MATLAB package described in [40] to perform the CQT. This does not provide a perfect reconstruction, as their later package described in [41] and another

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

similar package in Python [49] does, but we chose this because of the superior visual qualities as discussed in Section 3.1.4. Since we are not yet attempting to reconstruct the audio, this choice will not harm our analysis.

The parameters for the CQT are chosen to be similar to those in [26] and [43]. We use 24 bins per octave, between 12 (as in [26]) and 36 (as in [43]), which allows for the detection of quartertones (there are 12 semitones in an octave). The minimum frequency is A0, midinote 21, the lowest note on most pianos. The highest frequency is selected to be 9 octaves above this (A9, midinote 129). We could not choose a much higher note since the sampling frequency is 44.1 kHz and we would pass the Nyquist frequency (discussed in Section 3.1.2).

We downsample the signal to be twice the maximum frequency using the MATLAB resample function, which uses an antialiasing FIR lowpass filter. We do this to reduce the dimensionality of the spectrogram data. The new samplerate is 28,160 Hz. Given that, further down the pipeline, we perform a random selection of frames (see Section 4.1.2.3 below), we note that this was an oversight. In future work, this step will be removed.

The Q value of the CQT is 34.1271 and is determined by the number of bins per octave we selected. This means that representation of each frequency will consider approximately 34 cycles worth of samples within a window. The widest sample windows are those concerning the lowest frequency of 27.5 Hz, and require windows of just over 1 second for the DFT. The DFT window size and hop size are determined by the package per frequency, and the resulting CQT matrix has a frame rate of approximately 1000 Hz. This is a much higher frequency than 21.53 Hz (as in [26]) and 31.25 Hz (as in [43]); we chose this so as to minimise the amount of temporal information available to the algorithm. When performing the FFT, the package uses the square rooted Blackman-Harris windowing function, which we do not change. Finally, we take the absolute value of the complex numbers to produce a magnitude spectrum.

For an example of a note having undergone the CQT, see Figure 4.1.

4.1.2.2 Note detection

The files contain silence at the beginning and end. We found that a simple and effective method for excluding frames of (near) silence was to select frames with a standard deviation of greater than 0.1. This number was chosen an ad hoc way, but we checked programmatically that using this value always selected a single, continuous section of each note. We visually inspected all the CQTs to check that the selection contained the bulk of the note. Note detection had the greatest impact on the piano notes, which, in the raw recordings, are allowed to resonate for approximately 30 seconds; the preprocessed CQTs are approximately 5 seconds after the note detection.

4.1.2.3 Sampling frames

From each note CQT we select 100 frames randomly. After the note detection, all notes have similar durations, so random sampling should produce a small but representative dataset. However, from the literature e.g. [9] and empirically by plotting the spectrograms, we can see that the inter note variance is low but is largest at the beginning and end. To put is another way, the attack phase and the decay phase of the note are much more variant that the sustained middle section. Our random sample will not optimally capture the most information about notes, especially for longer notes, from which it may select fairly uniform frames. Since our evaluation set contains similar frames, we are in danger of being overconfident about predictions. To counteract this problem, as a final test, we predict all the frames of the evaluation notes i.e. not just the random sample of frames in the evaluation split.

4.1.3 Training, validation, and evaluation splits

Initially, we divided the frames *randomly* into 3 splits for training, validation, and evaluation. However, this resulted in near perfect instrument classification and note regression on the evaluation split by a simple nearest neighbour model. This indicated the variance within notes was low. Thus, to establish whether our models can generalise well, we insist that the pitch-instrument pairs are unique to each split e.g. if frames from Cello playing E♭4 are contained in the training split, there are no such frames in the validation or test splits. We achieve this by randomly splitting, checking if the conditions are met, then repeating if they were not.

The training, validation, and evaluation splits contain 40%, 10%, and 50% of the notes respectively. The purpose for this, seemingly quite harsh, split was to help detect overfitting. In particular, we want to guard against models associating a specific pitch with an instrument: there are a few f0 pitches within the training set for which there is only one instrument. A visual representation of the split is shown in Figure 4.3 and details of the number of notes per split are contained in Table 4.1.

4.2 Dimensionality reduction and visualisation

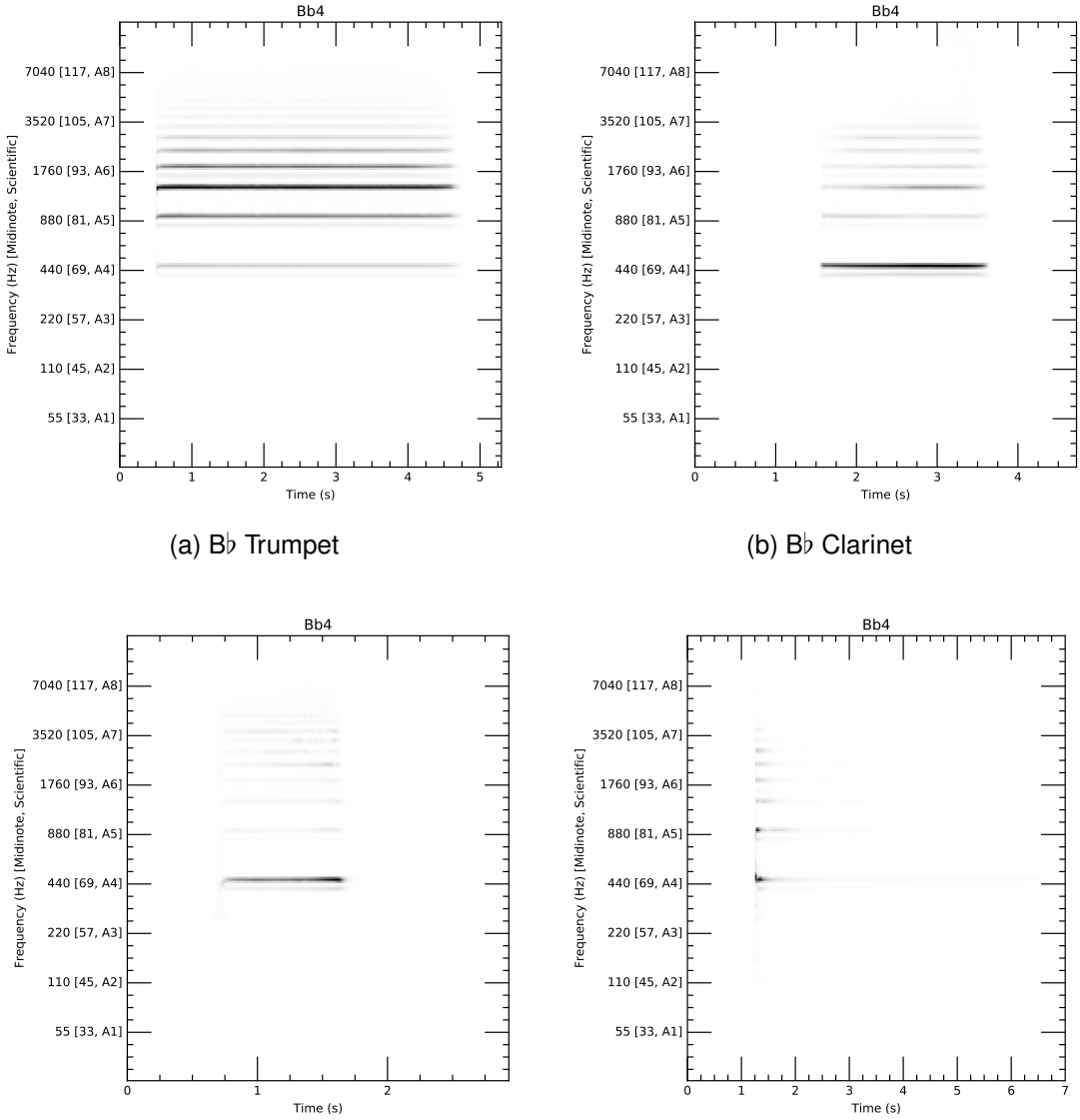
Before outlining the experiments, we provide some visualisations to give: an example of the input data, a view of the instrument ranges, and an initial indication that the data space is not trivially separable. We plot: some example CQT matrices in Figure 4.1 and Figure 4.2, the distribution of instruments within the data splits in Figure 4.3, and show the result of a simple dimensionality reduction using PCA in Figure 4.4.

We find that the CQTs for individual pitches have significant variance both between *and* within instruments. We were surprised to see the amount of variation within instruments, an example of which is shown in Figure 4.2. This initial visualisation indicates that the relationship between the relative strength of a note's harmonics and the instrument is clearly dependent on the f0 pitch.

	train	valid	eval
Bassoon	16	4	20
BbClarinet	18	2	18
BbTrumpet	13	4	19
Cello	18	7	21
DoubleBass	20	4	20
EbAltoSaxophone	9	5	18
Flute	11	3	15
Horn	13	3	28
Oboe	14	2	14
Piano	22	6	36
TenorTrombone	9	6	18
Tuba	17	1	19
Viola	14	4	22
Violin	10	2	21

Table 4.1: The number of notes contained in each split for each instrument. The split was performed randomly by note (not frame) and ensured that all instruments were contained in each split

Finally, we plot the standard deviation of the frequency dimensions in Figure 4.5. The purpose of this is for selecting a logical Gaussian noise parameter, and also to see where data is most variant with respect to each frequency dimension.



(a) B♭ Trumpet

(b) B♭ Clarinet

(c) Violin (arco on the G string)

(d) Piano

Figure 4.1: **The CQT matrix of four different instruments playing the same f0 pitch** (B♭4, midinote 70): This shows there is much inter instrument variance but, especially in the case of the Trumpet, very little intra note variance i.e. the magnitude of each harmonic in the notes does not change much after their onset. The dimensions of each matrix are 216×1000 per second. The input data for our models corresponds to *individual columns* of these matrices. See Figure 4.2 for an example of intra instrument note variance.

Figure 4.2: **The full range of B♭ Trumpet notes preprocessed by the CQT**: This shows that there is also much inter note variance within a single instrument i.e. there is clearly a difference in the relative harmonic strengths for different f0 pitches. This indicates that models will need to appreciate the register the instrument is playing in if they are to accurately classify them. Note also that there appears to be a strong frequency present throughout nearly all the notes. This could be a resonant frequency of the instrument itself as opposed to that of the column of air within the instrument. See Figure 4.1 for an example of inter instrument variance.

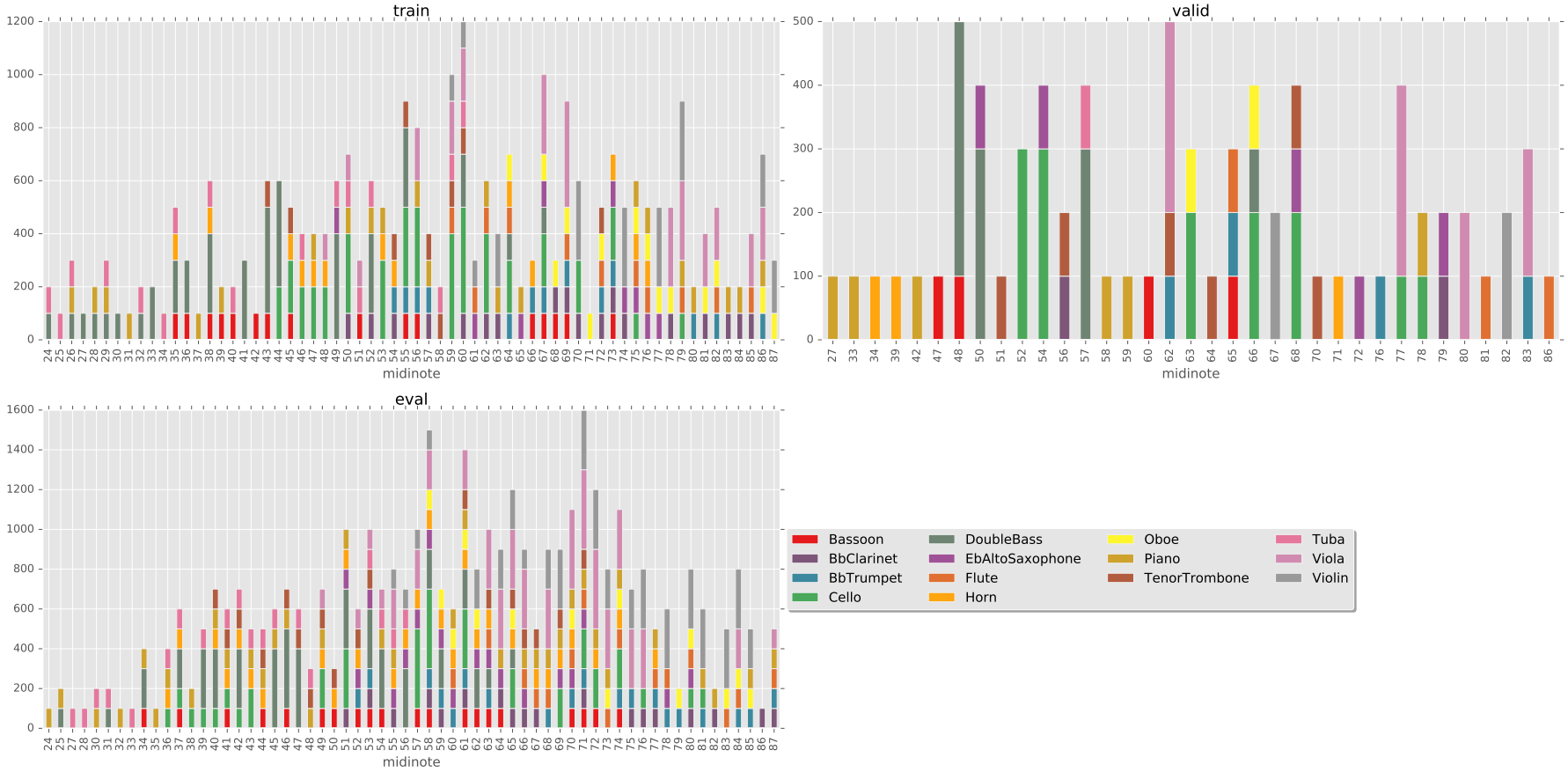


Figure 4.3: **A visual representation of the frame level data splits**: The stacked bar charts show midinote against number of frames. This is useful to provide the reader with an intuition about the split sizes and instrument ranges.

4.3 Experiment 1: Determine if there is signal in the magnitude of relative frequencies

In this experiment, we assess how easy it is to classify the instrument and to regress the pitch of a note using only the magnitude of frequencies relative to the f0 of the note. The models used are with mostly default parameters. The point is not to investigate too deeply, but simply to gain an initial insight, compare results with Experiment 2, then move on to train more complex models. In Experiment 2 we will use absolute frequency magnitudes.

To exclude any absolute frequency information (other than that contained within each of the harmonics e.g. their widths and magnitudes), we perform one additional preprocessing step to the data: selecting from 2 bins below the f0 of each note to 84 bins above the f0 (this covers 11 harmonics which span $\log_2(11)$ octaves ≈ 83.03 bins). Since the CQT has 24 bins per octave, the resulting preprocessed data are vectors of size $\texttt{ceil}[24\log_2 11] + 2 = 86$. We include the bins below the f0 as we observed that some notes have nonzero values here. This could be due to the CQT process i.e. 'leakage', or due to genuine vibrations of the instrument.

4.3.1 Instrument classification

We compare the performance of some 'out-of-the-box' classifiers implemented with Scikit Learn [34] and nolearn [33] (an abstraction of Lasagne [12] and Theano [46]). We choose to use:

nearest neighbours – to check the data space is sufficiently variant

logistic regression (one versus rest) – to indicate whether there is a linear relationship between relative frequency magnitudes and outcome

naive Bayes – to indicate whether instruments can be classified considering frequencies independently and to get a simple generative model of the input data

random forest with 500 trees - to see how well a competitive out-of-the-box classifier can perform

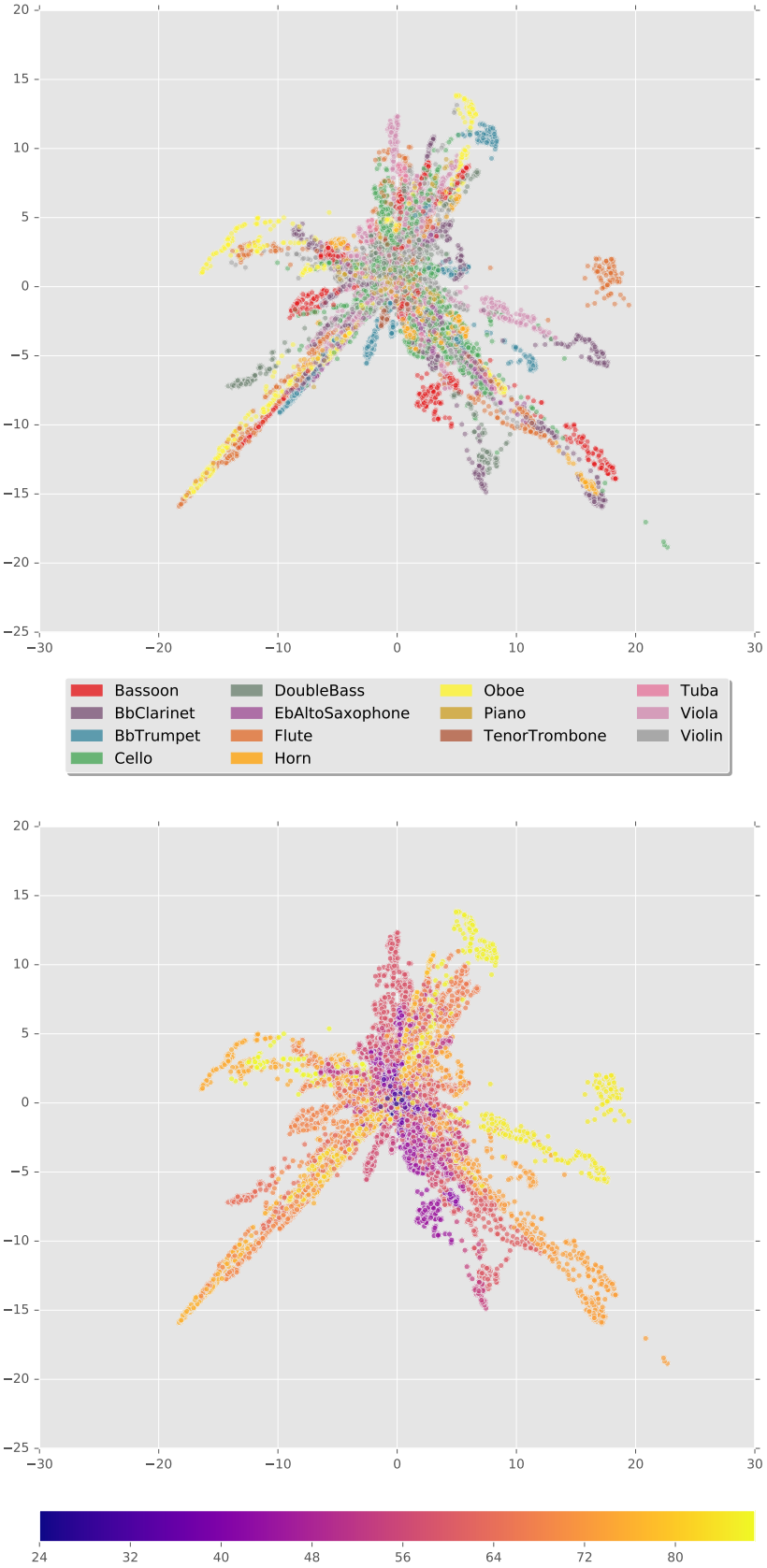
three neural networks with simple architectures – to see if the task is trivial without tuning

For the neural networks we test three simple architectures:

1. 1 hidden layer with 1000 nodes
2. 3 hidden layers with 100 nodes each
3. a CNN, which we refer to as CNN1-inst, with two convolutional layers with 100 filters of sizes 5 then 20 followed by a hidden layer of size 100

All neural network layers use a rectified nonlinearity, except for the output layer, which uses a softmax. We use a categorical cross-entropy loss and train the network with

Figure 4.4: **PCA visualisation of CQT frame data**: We perform PCA on the CQT frame data and plot the first two components. The top plot shows the distribution of instruments and the bottom the distribution of f0 pitches for the notes. We can see some definite groupings of instrument and f0 pitch tends to be lower towards the centre of the cluster. This shows that, whilst there is some structure to the data, the relationship between frame data and f0 pitch and especially instrument is not simple.



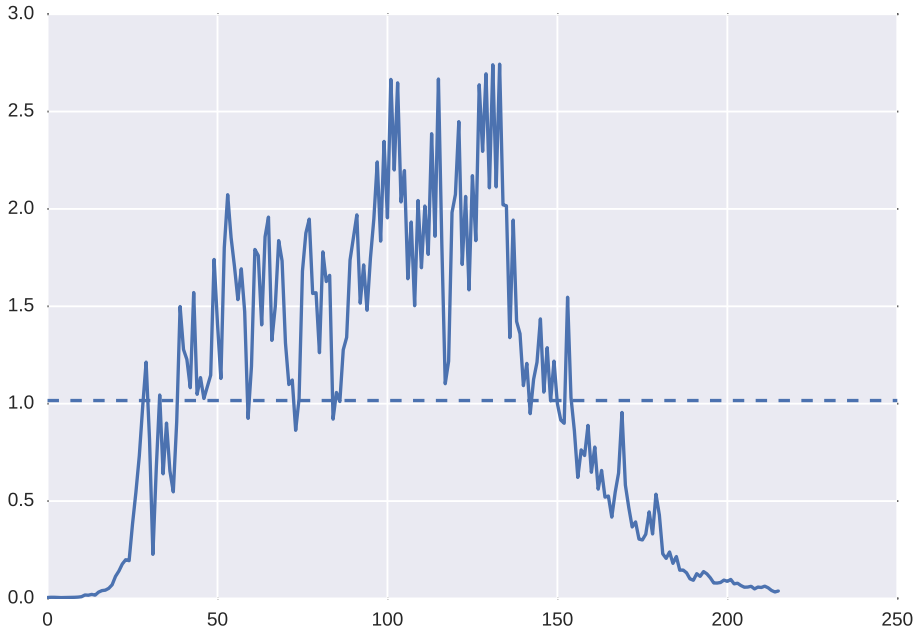


Figure 4.5: **Standard deviation within frequency bins**: Each of our data points is a vector with each dimension representing the amplitude of a frequency. We plot the standard deviation over the dataset for each dimension i.e. frequency. The lowest frequency is in dimension 0. The purpose for this is for selecting an appropriate noise parameter and to check that the data is variant within instrument ranges.

adam optimisation (with lasagne layer default values) and a batch size of 128. These choices are largely an ad hoc manner since the goal here is just to check that a function cannot be learned trivially.

We train all the classifiers on the training split and report the accuracy of the trained classifiers on the evaluation set. Since we do not attempt to fit parameters, the validation set is not used here.

4.3.2 F0 pitch regression

Similarly, we use 'out-of-the-box' regressors and choose to use:

nearest neighbours – to check the data space is sufficiently variant

linear regression – to indicate whether there is a linear relationship between relative frequency magnitudes and pitch

Bayesian ridge regression – to indicate whether regularisation and adding priors could be useful

random forest with 500 trees – to see how well a competitive out-of-the-box regressor can perform

three neural networks with simple architectures – to see if the task is trivial without tuning

The parameters of the three neural networks are exactly the same, except that now we are performing regression, so the loss is mean squared error and the output layer is a single linear unit. We refer to this adapted CNN as CNN1-f0

4.4 Experiment 2: Determine how performance changes with absolute frequency information

In this experiment we train identical models to those described in Experiment 1, except we do not do the extra preprocessing step i.e. we train on the full spectrum data. In this way, we see how performance of the models changes with the additional information of the absolute frequency magnitudes. For clarity, we refer to the CNN as CNN2-inst.

Given the same amount of data but an increased feature space, unless we are adding new and valuable features, it is natural to assume that performance would decrease. Since we expect absolute frequency information to be very revealing for pitch, we expect f0 pitch regression to improve. Conversely, we expect the instrument classification will suffer for the simple models such as nearest neighbour and linear/logistic regression. The additional preprocessing step from Experiment 1 could be seen as having performed an attention step on behalf of these models.

The input data are full time slices of the CQT: vectors of size 216 representing the magnitude of frequencies within a short window centred at a given time.

The instrument classification and f0 pitch regressors are identical to those used in Experiment 1 (bar the neural networks having a larger input layer). We refer to the CNN as CNN2-f0. Particularly in the case of the simple CNN, this could be seen as an unfair comparison given the data is of a different size. We are not too concerned at this point given that, for most of the models, the predictive power is about the same. This design should be sufficient to perform a simple comparison between the results of Experiments 1 & 2.

4.5 Experiment 3: Designing a CNN for instrument classification

We find that the CNN2-f0 architecture is capable of doing f0 pitch regression trivially, but for CNN2-inst, instrument classification is much harder (full results given in Chapter 5). The aim of this experiment is to train a CNN architecture, which we refer to as CNN3, using varying techniques with the goal of outperforming the Random Forest from Experiment 2 at the task of instrument classification.

It's worth emphasizing that no effort has gone into tuning the Random Forest, so we will not be making claims about the superiority of CNN3; successfully outperforming it implies only that it is possible to learn features for instrument classification (which we can then analyse). We use the Random Forest as a baseline as it attained the highest accuracy and because we found no exact duplicate of our experiment in the literature. Experiments from the literature used a mixture of different datasets, predict on a note level (not a frame level), and do not detail their data splits sufficiently to be reproduced. This issue is discussed further in Section 2.1.1. The random forest attains accuracies of around 85% which are similar to 90% accuracies reported in the literature (see Table 2.1).

Our first observation is that CNN2-inst is overfitting badly to the training data in Experiment 2. An example training plot is shown in Figure 4.6. We investigate the following methods for regularising the weights and improving the training:

- L2 regularisation
- Adding Gaussian noise to the input
- Applying dropout
- Applying batch normalisation
- Using pooling layers after convolutions
- Early stopping
- Augmenting the data

We give much more detail into the merits and reasons for use of these techniques in Section 3.3.

For the architecture of CNN3, we opt to use multiple convolutional layers. This choice was inspired by the design of Nouri's MNIST CNN[2], and Dielman's CNN for music

[2] http://danielnouri.org/notes/2014/12/17/using-convolutional-neural-nets-to-detect-facial-ke

	name	size		name	size
0	input	1x216	13	conv3_pool	32x18
1	input+n	1x216	14	conv3+d	32x18
2	input+d	1x216	15	conv4	64x6
3	conv1	8x205	16	conv4_batchnorm	64x6
4	conv1_batchnorm	8x205	17	conv4_pool	64x3
5	conv1_pool	8x103	18	conv4+d	64x3
6	conv1+d	8x103	19	hidden1	100
7	conv2	16x92	20	hidden1+d	100
8	conv2_batchnorm	16x92	21	hidden2	100
9	conv2_pool	16x46	22	hidden2+d	100
10	conv2+d	16x46	23	hidden3	100
11	conv3	32x35	24	hidden3+d	100
12	conv3_batchnorm	32x35	25	output	14

Table 4.2: **Layer sizes for training configuration 5 of CNN3, outlined in Experiment 3**: This details the layer size output for the network with dropout (+d layers), Gaussian noise (+n layers), batchnorm (_batchnorm layers), and pooling layers (_pool layers) for CNN3. In all other configurations the layers are in the same relative place and the architecture is the same. See Figure 4.7 for a schematic for the network.

tagging in Chapter 5 of their PhD thesis [11]. For a visual representation of CNN3 see Figure 4.7 and for a nolearn style table of the layers for configuration 5 below see Table 4.2. The first convolution is of size 12 which corresponds to half an octave. The convolution kernels for all four layers are of approximately size 12 and we perform max-pooling after each feature map, which downsamples by a factor of 2. This results in kernels that are functions of frequencies spanning $\frac{1}{2}$, 1.1, 2.4, and 6.2 octaves respectively.

Using max-pooling and a small number of feature maps at the first layer helps the network to generalise better: it only has a small number of blocks from which to build. Using rectified linear units as non-linearities can also regularise the network as some nodes can 'die' (the activation prior to the non-linearity always returns < 0).

We train the network with five configurations:

1. L2 normalisation with respect to all weights
2. Dropout added after all layers
3. Dropout plus Gaussian noise added to the input
4. Dropout plus batch normalization after convolutions
5. Dropout, Gaussian noise, plus batch normalisation

We experimented with L2 and found that, with a lambda value of much more than 0.35 the network would quickly converge to a suboptimal place, with a lambda value of less than 0.25 it would overfit dramatically. We don't report figures for this since it

was found not to be as useful in the presence of dropout; additionally, it worsened the solution when used in conjunction with dropout and other methods.

For the dropout parameters, we found that if values were too small the network would overfit. We set the parameters such that the level of dropout is increased further along in the network.

We choose the following dropout parameters in an ad hoc manner (abbreviations follow terminology in Figure 4.7):

- Input: 0.1
- C1: 0.2
- C2: 0.3
- C3: 0.4
- C4: 0.5
- F5: 0.5
- F6: 0.6
- F7: 0.7

For the Gaussian noise standard deviation parameter, we observed the standard deviation of each dimension (result shown in Figure 4.5), finding the average to be about 1. We tried using noise with a standard deviation of 0.1 but found this to very negatively impact the performance. We found that a standard deviation of 0.01 produced positive results.

Batch normalisation is implemented between the layer output and the nonlinearity for all the convolutional layers.

Finally, the early stopping approach we take is to continue training until there has been no improvement in the validation loss for 200 epochs. We then revert to using the weights at the epoch with the best validation accuracy.

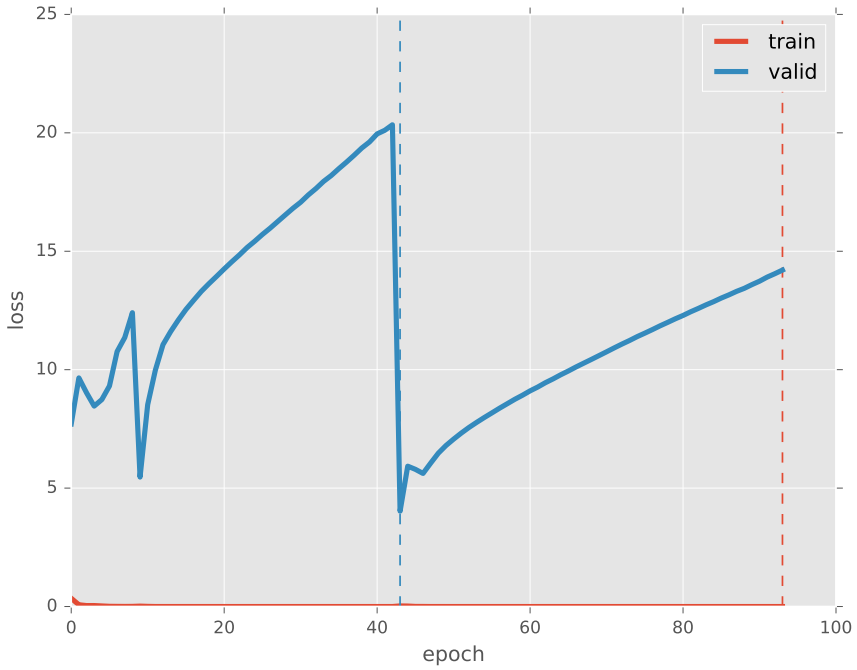


Figure 4.6: **An example training plot for the CNN2-inst in Experiment 2**: We find that, for instrument classification on the full frequency frame data, CNN2-inst immediately overfits; the optimisation algorithm does well at minimising the training error but does so by setting the parameters to values that are specific to the training set i.e. the network does not generalise. This motivates the use of regularisation techniques, and considerably fewer learnable parameters, explored in Experiment 3.

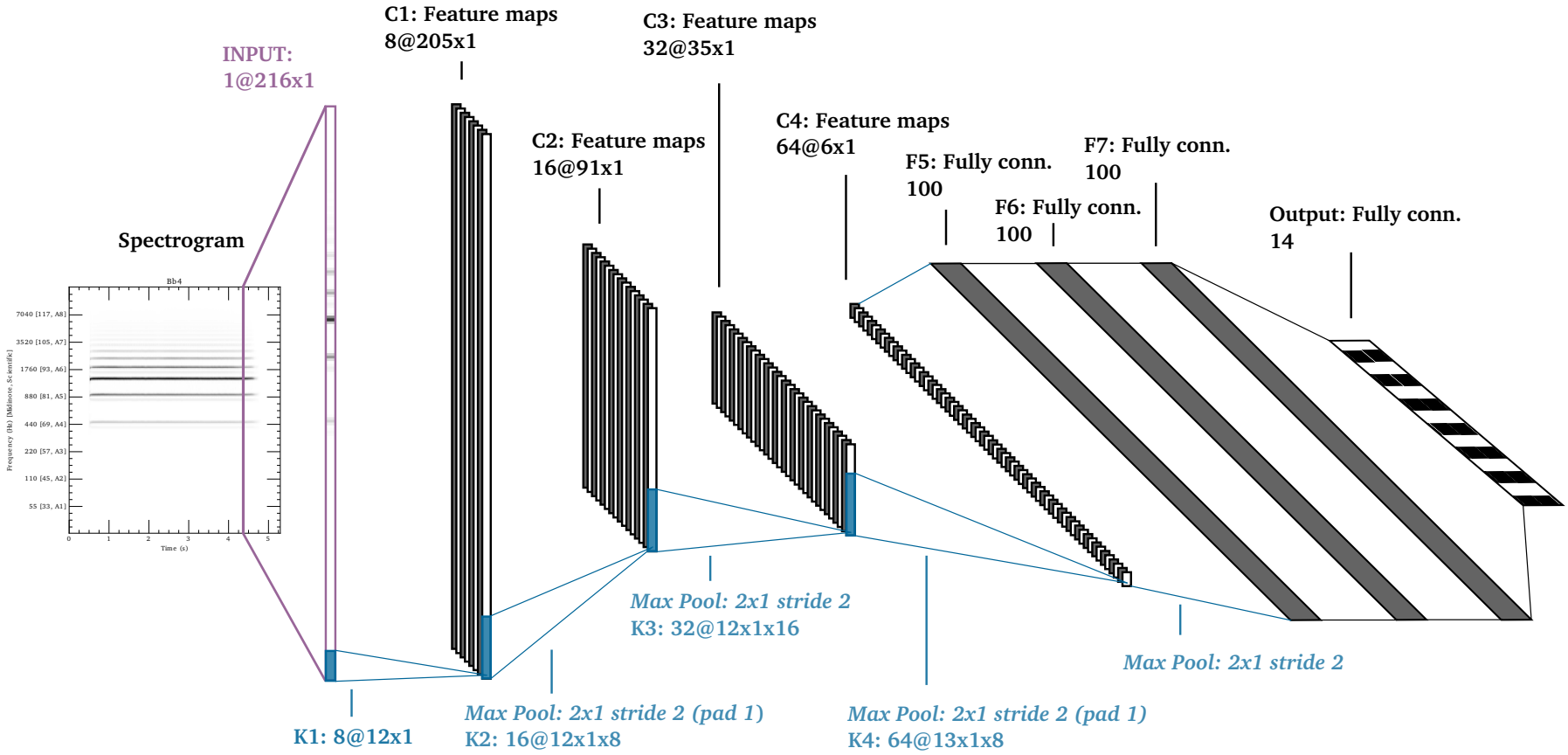


Figure 4.7: **Design of CNN3 outlined in Experiment 3**: CNN3 is composed of 4 convolutional layers followed by 3 fully connected hidden layers and one finally one fully connected output layer with a softmax nonlinearity. After each feature map we implement max-pooling. The sizes of the kernels can be described in terms of musical octaves. This helps the reader to think about the types of features it is possible for each to learn. Each feature map carries forward the information from the input layer $= 9$ octaves. The kernels K1 – K4 span $\frac{1}{2}$, 1.1, 2.4, and 6.2 octaves respectively. For reference, there are at least 2^x harmonics in any given x octaves above the f0 frequency. The feature maps and kernels (accounting for max pooling) are to scale with respect to the input.

4.5.1 Processing time

We perform the network training using a Tesla K40m GPU. The data is reasonably small so does not demand much memory (about 300 MB). Training times are of the order of 10s per epoch and most configurations will converge within 200 epochs leading to training times of approximately 30 minutes per experimental configuration.

For comparison, performing the experiments on a CPU (Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz) result in a slow down of approximately 5 times i.e. experiment training takes approximately 3 hours.

4.5.2 Data augmentation

After performing the comparison of these parameters, we repeat the experiment with all 5 configurations again with augmented data. We follow one of the procedures outlined in [38].

We hypothesize that nearby pitches have similar structure and augment the data by shifting the vectors up and down by 1 and 2 frequency bins (padding zeros at the top or bottom as required). This is equivalent to shifting up and down by a quarter tone and a semitone.

Since this increases the size of the data, this impacts the training times. We find that, using the Tesla 40m GPU, training times are of the order of 30s per epoch and most configurations will converge within 200 epochs leading to training times of approximately 2 hours per experimental configuration.

Chapter 5

Results

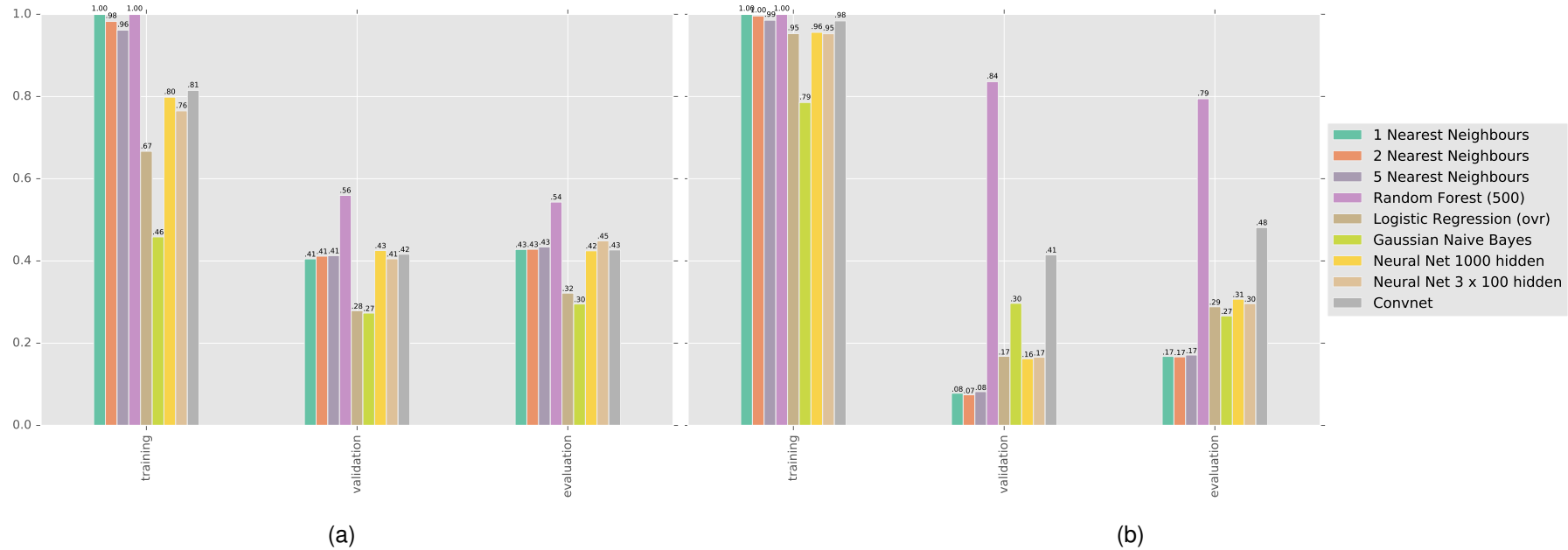
5.1 Summary

We find that frequency magnitudes relative to the f0 are enough to produce frame instrument classifications with an accuracy of 54% using an 'out-of-the-box' random forest with 500 trees. That increases to 79% when given information about absolute frequency. If we augment our training data, by shifting it up and down 1 and 2 frames, we can increase that accuracy to 85%.

CNN3, outlined in section 4.5, attains a classification accuracy of 90% on the same data. Whilst the random forest uses approximately 6,000,000 parameters, whilst CNN3 uses fewer than 76,000. With two orders of magnitude fewer parameters, this results in a significantly less complex model upon which to build for future work.

To compare as best we can with studies from the literature (issues discussed in Section 2.1.1), we also provide a note level accuracy and find that here CNN3 performs even better comparatively, predicting 95% of the notes correctly (94% if we average by class). The mistakes the network makes are very similar to those shown within the literature; for example, Saxophone is the most difficult to classify and is often mistaken for Viola.

Finally, we analyse where CNN3 makes mistakes with respect to time relative to the onset of the note, and the pitch of the note's f0. We find that, as is to be expected from the literature, our classifier makes mistakes at the beginning and the ends of notes more frequently. This to make a similar conclusion to [47]: in order to make significant improvement to instrument classification, one must use temporal information. However, unlike [47], our results show that high classification accuracy can be obtained without explicit temporal features. We uncover some potential issues with our training data, which highlight both ways to improve performance further in this task, and that there is room for improvement in CNN3's ability of generalise instrument detection across pitches.



(a) (b)

Figure 5.1: **Experiment 1 & 2 instrument classification accuracies**: We show the results of the instrument classification task above. In Experiment 1 (shown in plot a) the data were preprocessed to be relative to the f0 frequency of each note (the CNN is CNN1-inst); for Experiment 2 (shown in plot b), we used the full spectrum such that absolute frequency information is available (the CNN is CNN2-inst). The random forest performs best in both cases. *N.B. Since the trees are grown to full depth, it is not unusual for a random forest to report 100% accuracy on the training data.*

5.2 Experiments 1 & 2: Relative and absolute frequency magnitudes

5.2.1 Instrument classification

Our first observation is that the Random Forest produces the best results 'out-of-the-box'. Given that it can model interactions and non-linearities, and that it is an ensemble method designed to have low generalisation variance, this is not particularly surprising. The accuracies are shown in full in Figure 5.1.

The random forest performs considerably better than the logistic regression. Whilst this is a slightly unfair comparison, it indicates that there are non-linear relationships between the frequencies at the harmonics and the outcome. Figures 5.2 & 5.3 provide some additional backing to this theory: the random forest relative importance plot shows frequencies at the location of the harmonics as most important, whereas the logistic regression weights are smallest at these locations. This indicates that the harmonic frequencies both provide useful information, and have some non-linear relationship with the instrument class.

5.2.1.1 Including absolute frequency information

Unsurprisingly we find that including absolute frequency information improves instrument classification for the random forest; we interpret the reason for this to be that it can leverage information about register being used e.g. if it detects low frequencies with high magnitude, it knows that the flute is an unlikely classification. This is supported by looking at the relative importance plot in Figure 5.4; this shows a fairly even importance across all frequencies.

Including absolute frequency information has the adverse affect on the nearest neighbour classifiers. Using the 5 nearest neighbours to classify each frame attains an accuracy of 43% using relative frequencies, which drops to 17% when absolute information is included. This suggests that:

(a) Experiment 1 shows us that, over the whole range of a given instrument, there exists a pattern in the relative strength of the harmonics that can be used to predict that instrument
(b) Experiment 2 shows us that, when comparing notes with the same f0 pitch, there exist significant similarities between instruments. For a window into these similarites, observe the classification errors presented in the confusion matrix in Figure 5.9

CNN2-inst performs better than both the other neural network architectures, but is considerably outperformed by the random forest. The fact that training accuracy is 98% whilst the evaluation is just 48% hints at the underlying problem: the model is severely overfitting. Indeed, as we see from the training plot in Figure 4.6, this is

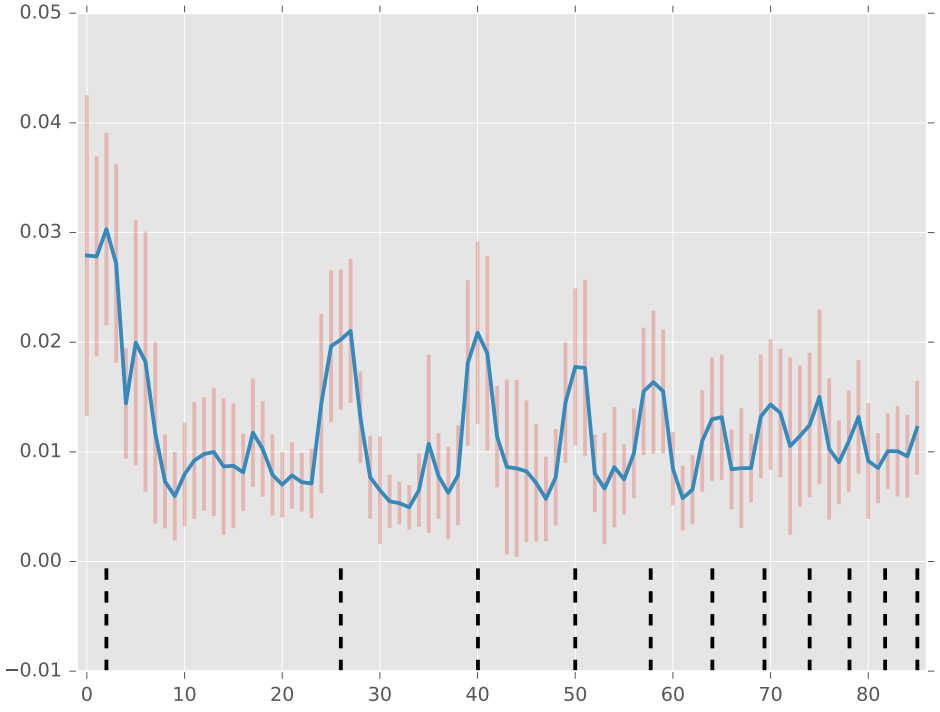
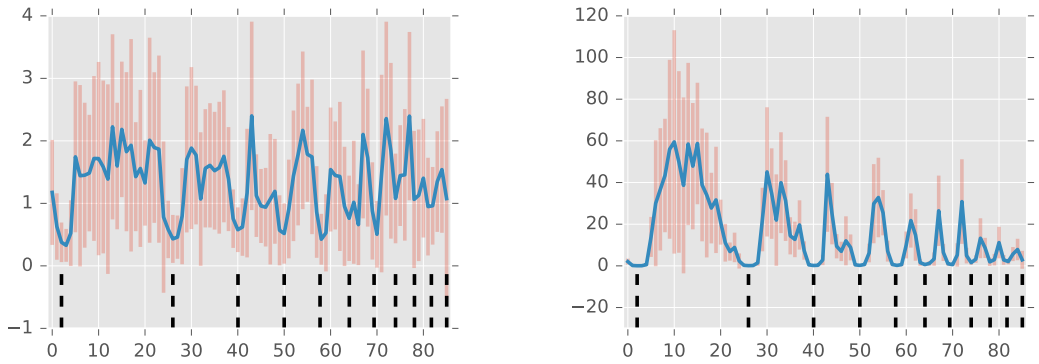


Figure 5.2: **Experiment 1 random forest relative importances**: We show the relative importance (y axis) of each of the frequency bins (x axis) used by the model for Experiment 1. We see that the most important features are where the harmonics are the strongest in the lower frequencies and that the upper frequencies all have similar importance. The red lines denote the standard deviation with respect to the trees in the forest (*N.B. this is not the same as the red lines in Figure 5.3*). The dotted black lines show the position where the natural harmonics would fall.



(a) (b)

Figure 5.3: **Experiment 1 logistic regression weights**: We show the mean absolute values of the weights (y axis) for each of the frequency bins (x axis) used by the model. Plot 5.3a shows the absolute values of the weights and plot 5.3b scales these values by the mean of the data. Since this is a one-versus-rest model, we are actually showing the mean values; the red lines show the standard deviation with respect to the 14 different instrument models (*N.B. this is not the same as the red lines in Figures 5.2 or 5.4*). The dotted black lines show the position where the natural harmonics would fall.

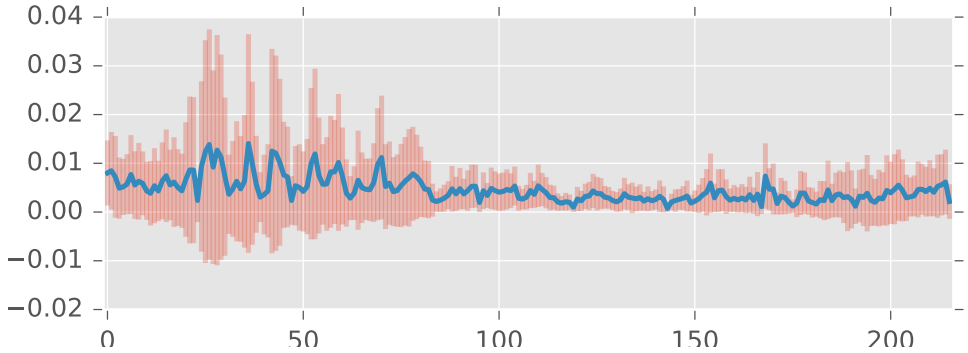


Figure 5.4: **Experiment 2 random forest relative importances**: We show the random forest relative importance (y axis) for the full range of frequency bins (x axis). The vertical red lines denote the standard deviation with respect to the trees in the forest. We observe that the importances are relatively even but that there are some spikes in the lower frequencies. This provides some indication that model may be overfitting. Some pitches, especially in the lower register, are played by only one instrument in the training data; this could well be what the model is learning i.e. if it sees, for instance, an A2 it predicts a Double bass.

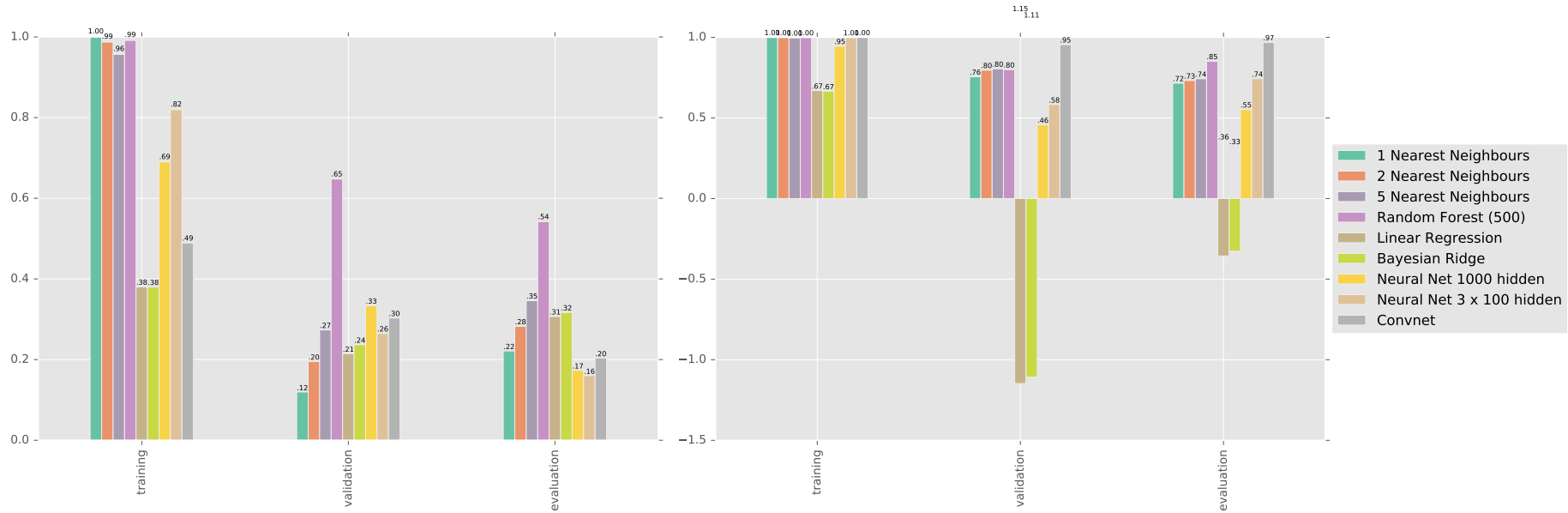


Figure 5.5: **Experiment 1 & 2 pitch regression R^2 scores**: We show the results of the note regression task. In Experiment 1 (shown in plot a) the data were preprocessed to be relative to the f0 frequency of each note (the CNN is CNN1-f0); for Experiment 2 (shown in plot b), we used the full spectrum such that absolute frequency information is available (the CNN is CNN2-f0). The R^2 score is plotted. The random forest performs best in both cases. *N.B. Since the trees are grown to full depth, it is not unusual for a random forest to report 100% accuracy on the training data.*

indeed the case. There are approximately 2 million parameters in this model; we find that we can reduce this dramatically to improve performance.

5.2.2 F0 pitch regression

Our first observation is that, because simple classifiers such as linear regression are able to predict the pitch of the note in Experiment 1, there is definitely signal within the relative frequencies about the pitch. Again, we see that, the random forest performs best out of the box. The accuracies are shown in full in Figure 5.5.

5.2.2.1 Including absolute frequency information

However, with the addition of absolute frequency information in Experiment 2, CNN2-f0 outperforms all models. CNN2-f0 attains an R^2 score of 0.97, with the random forest attaining 0.85. To recap Section 4.4, CNN2-f0 has two convolutional layers, the second of which has 100 filters which model octave size patterns. Since CNN2-f0 performs convolutions over the length of the vectors, it is able to model the position of these shapes. This is clearly enough to identify the pitch of each note.

The nearest neighbour regressors *improve* when given absolute frequency information; this is the opposite of what happened for instrument classification. This leads us to conclude:

(a) Experiment 1 shows us that, over the whole range of instruments for a given pitch, there exists a pattern in the relative strength of the harmonics that can be used to predict that pitch
(b) Experiment 2 shows us that, when comparing notes with the same f0 pitch, there exist significant similarities between instruments. For a window into these similarities, observe the classification errors presented in the confusion matrix in Figure 5.9

Conclusion (b) is the same for instrument classification and pitch regression.

Since CNN2-f0 attains near perfect prediction at this task, we now turn to the task of instrument classification only.

5.3 Experiment 3: Designing a CNN for instrument classification

Our attention now turns to the task of improving CNN2-inst, the CNN for instrument classification given absolute frequencies. We re-design the network architecture when creating CNN3 to dramatically reduce the number of parameters by reducing the number of feature maps but increasing the number of layers. The network design is shown

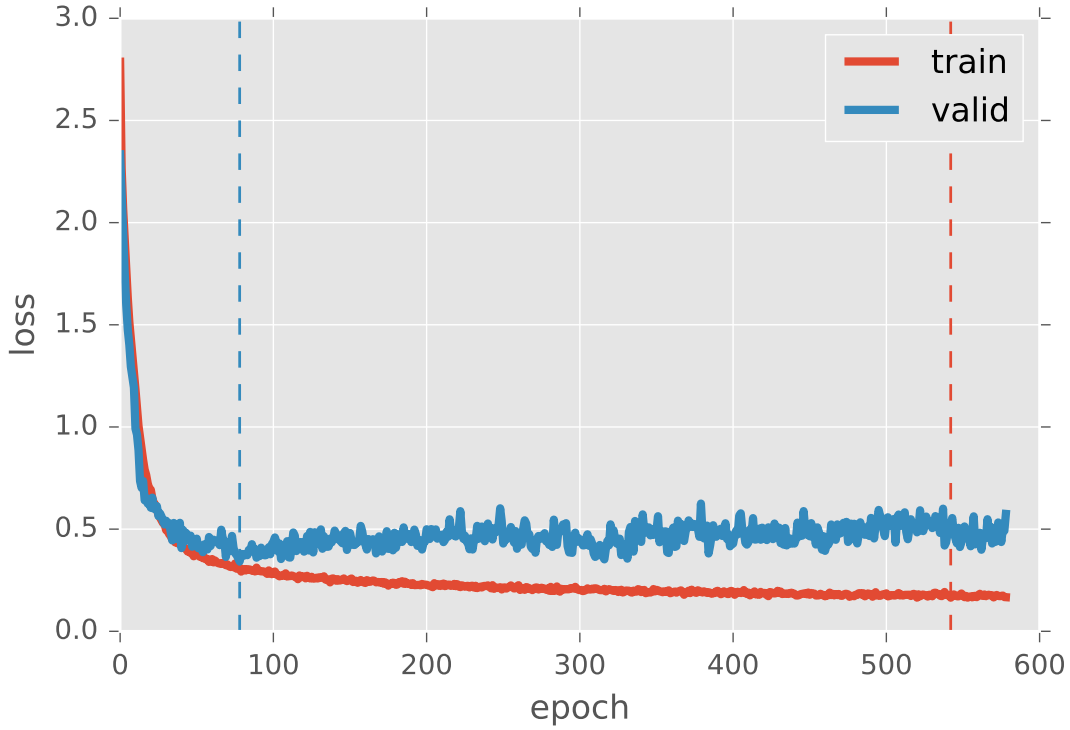
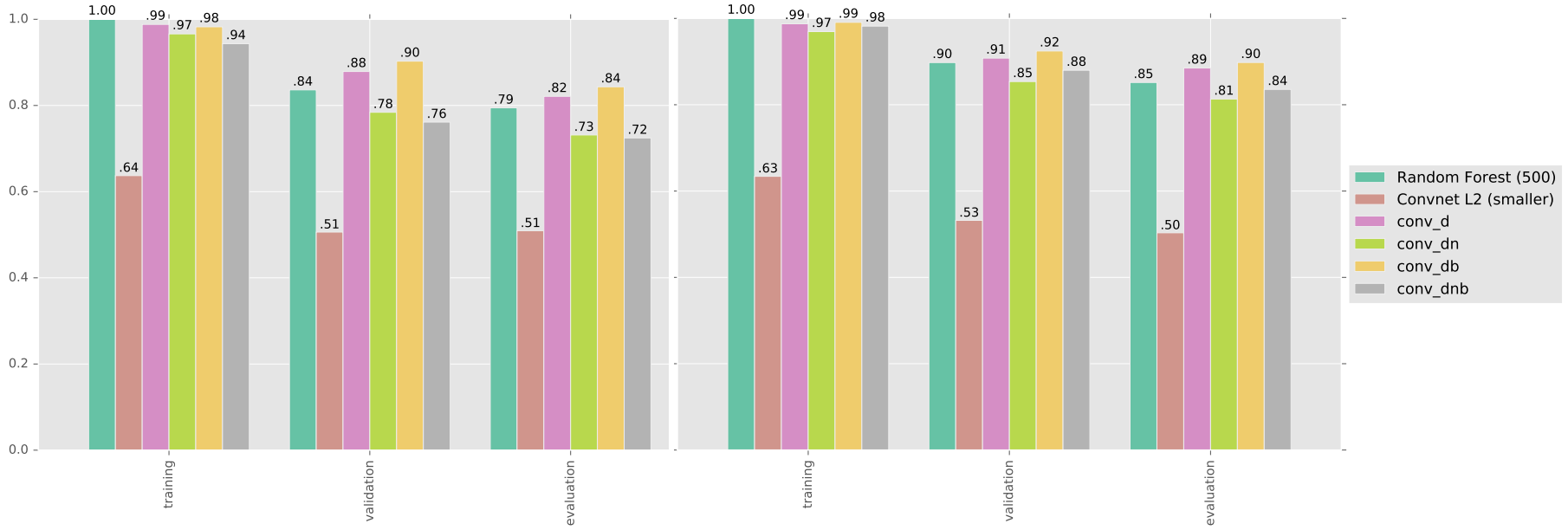


Figure 5.6: **An example training plot for CNN3 in Experiment 3**: The above plot shows the progression of the cross-entropy loss for CNN3 trained with dropout and batch normalisation ('conv_db' in Figure 5.7). This network was trained *without* data augmentation. We see much improvement over the previous network, CNN2-inst, outlined in Experiment 2, whose training plot is shown in Figure 4.6: the validation error now tracks the training error for the first 50 epochs.

in Figure 4.7. The resulting model has fewer than 76,000 parameters, which is down from over 2,000,000 in Experiment 2.

The results for the exploration of different regularisation techniques, *without* data augmentation, are shown in Figure 5.7a. Some things are immediately clear: L2 normalisation is not effective, but using dropout certainly is. Using dropout alone we attain an accuracy of 82%; with the addition of batch normalisation, this increases to 84%. We find the addition of Gaussian noise only hinders the learning.

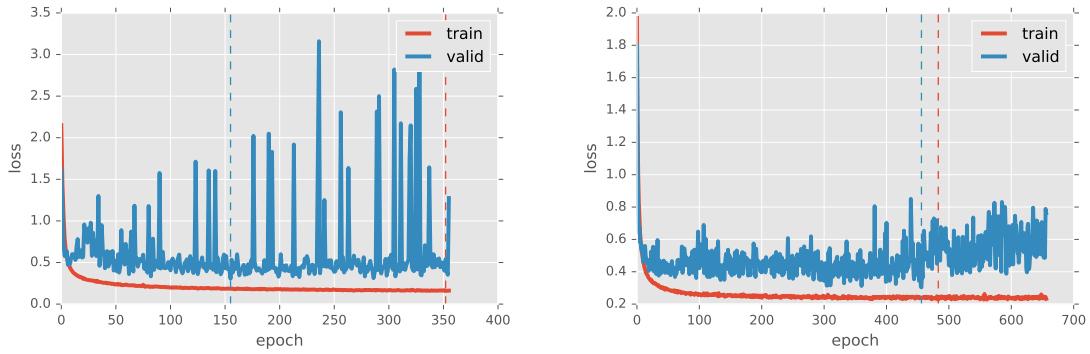
The training and validation errors are still greater than evaluation error. This implies that the classifier is still somewhat overfitting. We show the training and validation loss against epoch number in Figure 5.6. There is a big improvement over the previous network: the validation error tracks the training error well for the first 50 epochs, at which point the validation curve flattens out. It's worth noting at this point that the validation set is somewhat limited, containing just a few notes for some instruments.



(a) *no* data augmentation

(b) *with* data augmentation

Figure 5.7: **CNN3 performance with different training strategies**: Plot (a) shows accuracies without data augmentation, and plot (b) shows with. The abbreviations in the legend stand for: _d with dropout; _dn with dropout and gaussian noise; _db with dropout and batchnorm; _dnb with dropout, noise, and batchnorm. The architecture of the convolutional network is the same for all 'conv_' models, but is reduced in number for 'Convnet L2' (to attempt to account for dropout). We find that augmenting the data increases accuracy by approximately 6 percentage points for all models (bar 'Convnet L2')



(a) with dropout + batchnorm

(b) with dropout only

Figure 5.8: **Experiment 3 data augmentation**: (a) shows the result of training CNN3 with batch normalisation and dropout; (b) shows the same network trained with just dropout. We note the somewhat erratic training curve. This problem did not occur with before data augmentation (as shown in Figure 5.6). In both cases, we see much more variation in validation error loss

5.3.1 Data augmentation

We augment the data by shifting the frame vectors up and down by 1 and 2 frequency bins i.e. we increase the dataset to 5 times its original size. This aids both the random forest and all training configurations for CNN3 by approximately 6 percentage points, as can be seen by comparing the plots in Figure 5.7.

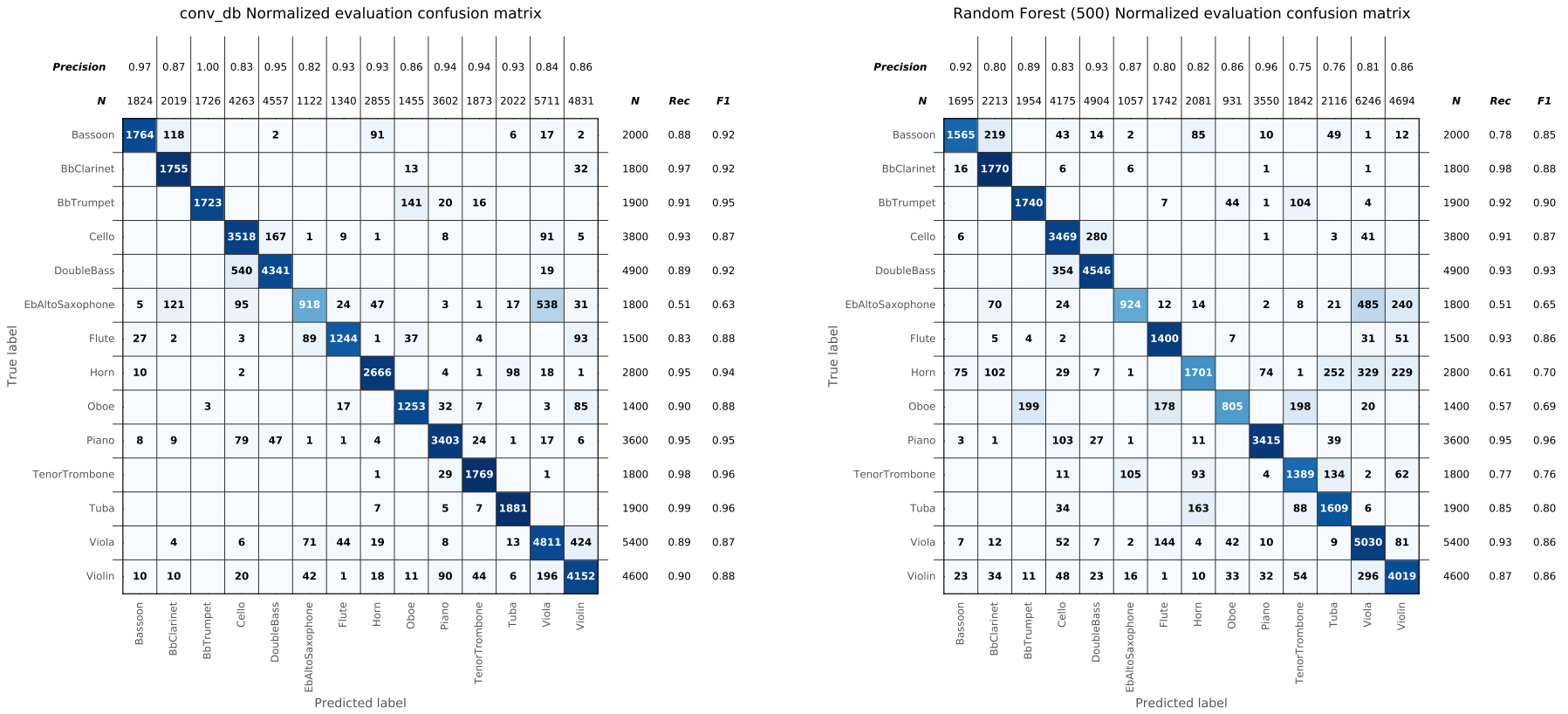
We note that the validation accuracy is somewhat erratic when using batch normalisation. This is not such an issue when we train using only dropout i.e. without batch normalisation, as shown in Figure 5.8b.

As without data augmentation, the best performing training strategy is to use dropout after all layers and batch normalisation after the convolutional layers: this configurations attains an evaluation accuracy of 90% (up from 84% without data augmentation).

5.4 Note level analysis

To compare with the literature, and to analyse how our classifiers might be improved with temporal information, we now discuss classifier performance on a note level. To perform note classification, we simply classify every frame of the note CQT, take the mean of the predicted probabilities, and select the class with the highest mean. We could, of course, use another classifier to model these predictions, but this would implicitly use temporal information. We found averaging the predicted probabilities performs better than averaging the frame classification by about 1 percentage point.

CNN3 with the best training strategy attains a note level accuracy of 95% (94% if we take the mean accuracy per class); by comparison the random forest achieves 88% (87% mean accuracy per class).



(a) CNN3

(b) Random forest

Figure 5.9: **Frame level confusion matrices**: Figure (a) shows the confusion matrix for the best performing training configuration for CNN3, (b) the random forest. For a higher level analysis, see the note level confusion matrices in Figure 5.16.

Full details of class level accuracies are given in Table 5.1. Figure 5.16 shows the note level confusion matrices with precision, recall, and F1 scores.

5.4.1 Where prediction fails

Figure 5.11 shows how prediction accuracies vary with time and pitch. As we noted in our original data analysis (Section 4.2), note beginnings and ends are the most variant within instrument classes. Indeed, we find that accuracy follows a smooth inverted U curve.

We also find the mean predicted probability of the correct class to be below the lower quartile in Figure 5.11a. This is because the classifier predicts very extreme probabilities i.e. it often predicts a class with 100% probability, resulting in an either completely correct or completely incorrect result.

We find that Saxophone is particularly hard to predict. Whilst this is also found in the literature (see [47]), our random split resulted in a disproportionately small number of Saxophone notes in our training split; the random forest also struggles to correctly classify saxophone, so it is possible this is a data issue. The CNN tends to predict most frames in some saxophone notes almost perfectly, but for some notes it classifies all but a few frames incorrectly.

In Figure 5.10 we plot the class predictions against midinote to see if there is any pattern with respect to pitch (see the frame level confusion matrix in Figure 5.9 for a full frame level breakdown). We note the poor performance in the register between midinote 53 and 59. The training split is shown in Figure 4.3; from this, we see there is indeed a gap in the training data between midinote 50 and 59 inclusive. We conclude, therefore that the poor performance could well be put down to a data issue; the model hasn't seen enough good examples of saxophone frames in that region.

Finally, we plot the predictions for an example note by the best CNN trained with (Figure 5.12) and without (Figure 5.13) data augmentation; likewise for the random forest (Figures 5.14 & 5.15 respectively). This gives a flavour of the difference between the classifiers, and hints at the cost of the data augmentation: some additional uncertainty in previously well classified examples. The note (Flute playing B♭4) is a good example of all the classifiers struggling to classify frames near the onset of the note.

(a) CNN3

instrument	nr	class_acc	pred_proba_acc
Bassoon	20	0.900	0.900
BbClarinet	18	1.000	1.000
BbTrumpet	19	0.947	0.947
Cello	21	1.000	1.000
DoubleBass	20	1.000	1.000
EbAltoSaxophone	18	0.556	0.556
Flute	15	0.867	0.867
Horn	28	1.000	1.000
Oboe	14	0.929	0.929
Piano	36	1.000	1.000
TenorTrombone	18	1.000	1.000
Tuba	19	1.000	1.000
Viola	22	1.000	1.000
Violin	21	0.952	0.952
MEAN	289	0.948	0.948
MEAN CLASS	14	0.939	0.939

instrument	nr	class_acc	pred_proba_acc
Bassoon	20	0.850	0.850
BbClarinet	18	1.000	1.000
BbTrumpet	19	0.947	0.947
Cello	21	0.905	0.857
DoubleBass	20	1.000	1.000
EbAltoSaxophone	18	0.611	0.611
Flute	15	1.000	1.000
Horn	28	0.643	0.643
Oboe	14	0.643	0.643
Piano	36	1.000	1.000
TenorTrombone	18	0.944	0.944
Tuba	19	0.895	0.895
Viola	22	1.000	1.000
Violin	21	0.810	0.810
MEAN	289	0.879	0.875
MEAN CLASS	14	0.875	0.871

(b) Random forest

Table 5.1: **Note level accuracy comparison**: Table (a) shows the note level accuracy of the best performing training configuration for CNN3, (b) the random forest. The abbreviations 'class_acc' and 'pred_proba_acc' indicate whether note predictions were created by taking the mean of the frame classifications, or predicted probabilities respectively. MEAN shows the mean accuracy by note, and MEAN CLASS shows the mean accuracy by instrument. CNN3 performs well on all instruments apart from Saxophone. For a more detailed analysis of where errors are made, see the confusion matrices in Figure 5.16.

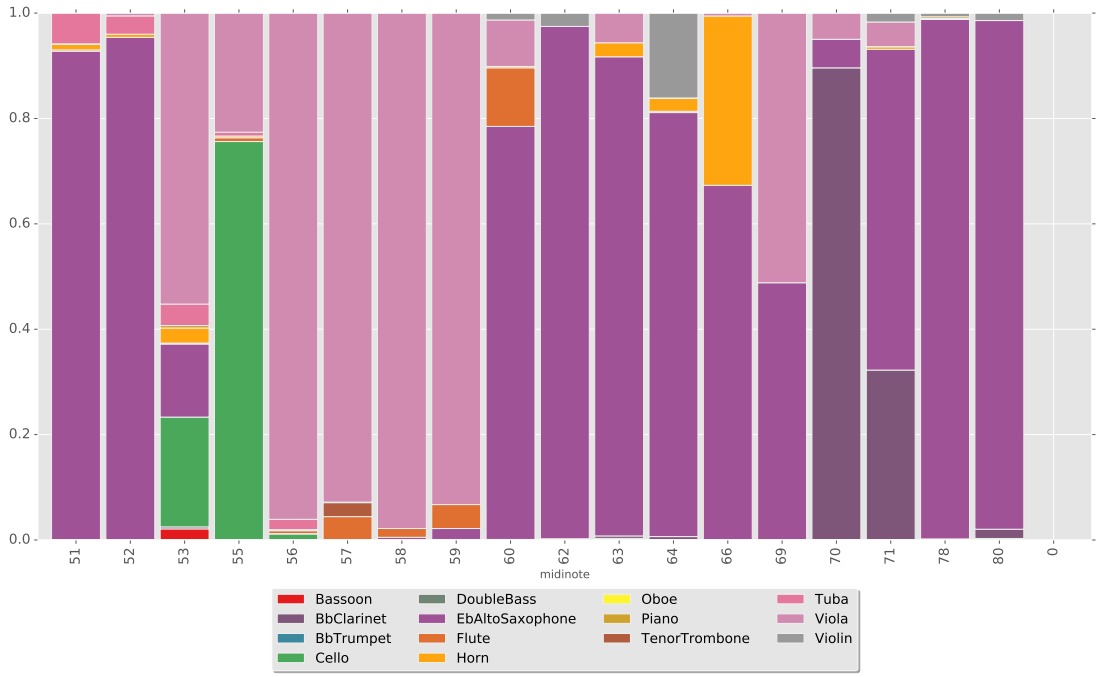
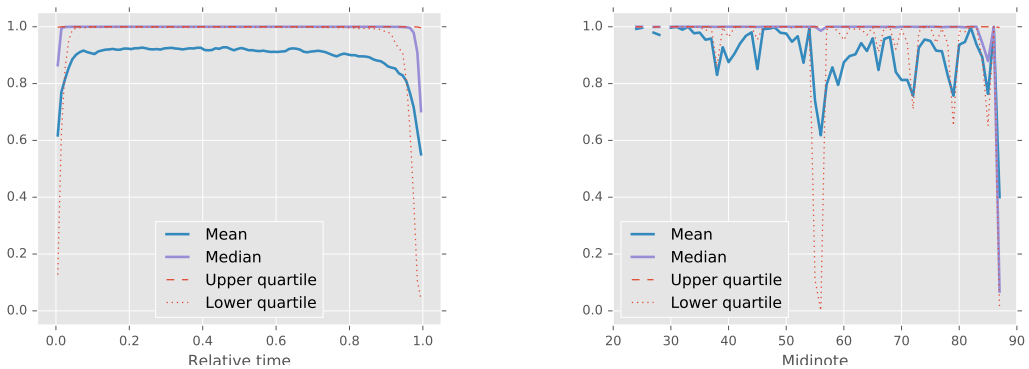


Figure 5.10: **CNN3's Saxophone misclassification errors with respect to pitch**: We plot the proportional class predictions made by the best performing training configuration for CNN3 for each Saxophone note in the evaluation set. The x axis shows the notes in order of pitch, low to high. We note a continuous gap in performance between midinote 52 and 59 inclusive. This is discussed in the text.



(a) (b)

Figure 5.11: **Analysing where prediction fails**: These plots show statistics of the predicted probabilities generated by CNN3 trained on augmented data in Experiment 3. Plot 5.11a shows the statistics against the time relative to the onset of the note. We see a clear pattern that the classifier is less confident at predicting the beginnings and ends of each note. From the literature we know this is expected: attacks and decays of notes often exhibit different characteristics from the sustained main part. Additionally we can see that the mean is typically lower that the lower quartile. This is as a result of there being many very high and very low probabilities. Plot 5.11b shows the statistics against the midinote i.e. the pitch of the f0 of the note. Statistics of the predicted (binary) class showed the same pattern. The is not such a clear pattern here though performance does seem to drop as pitch increases. The dip between midinote 50 and 60 is partly due to the poor saxophone classification; this is detailed in Figure 5.10.

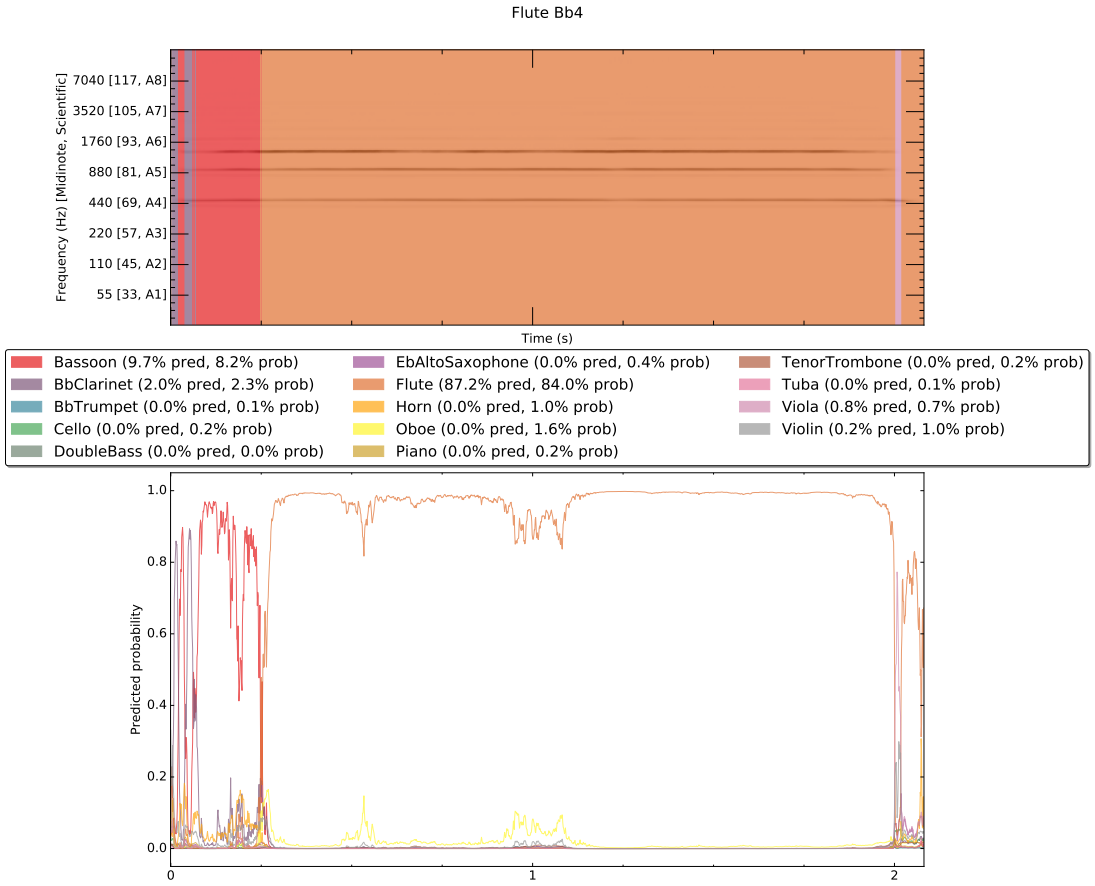


Figure 5.12: **CNN3 (*not* trained on augmented data) note prediction**: An example frame-by-frame classification of an evaluation note by CNN3. The top plot shows the spectrogram of the note, overlaid with a colour representing the predicted class for each frame. The bottom plot shows the classifier's predicted probability for every class. The classifier struggles to predict frames near the onset, but performs well after about 0.25 seconds. It is interesting to contrast this prediction with the random forest (shown in Figure 5.14), which mistakes the onset for Violin.

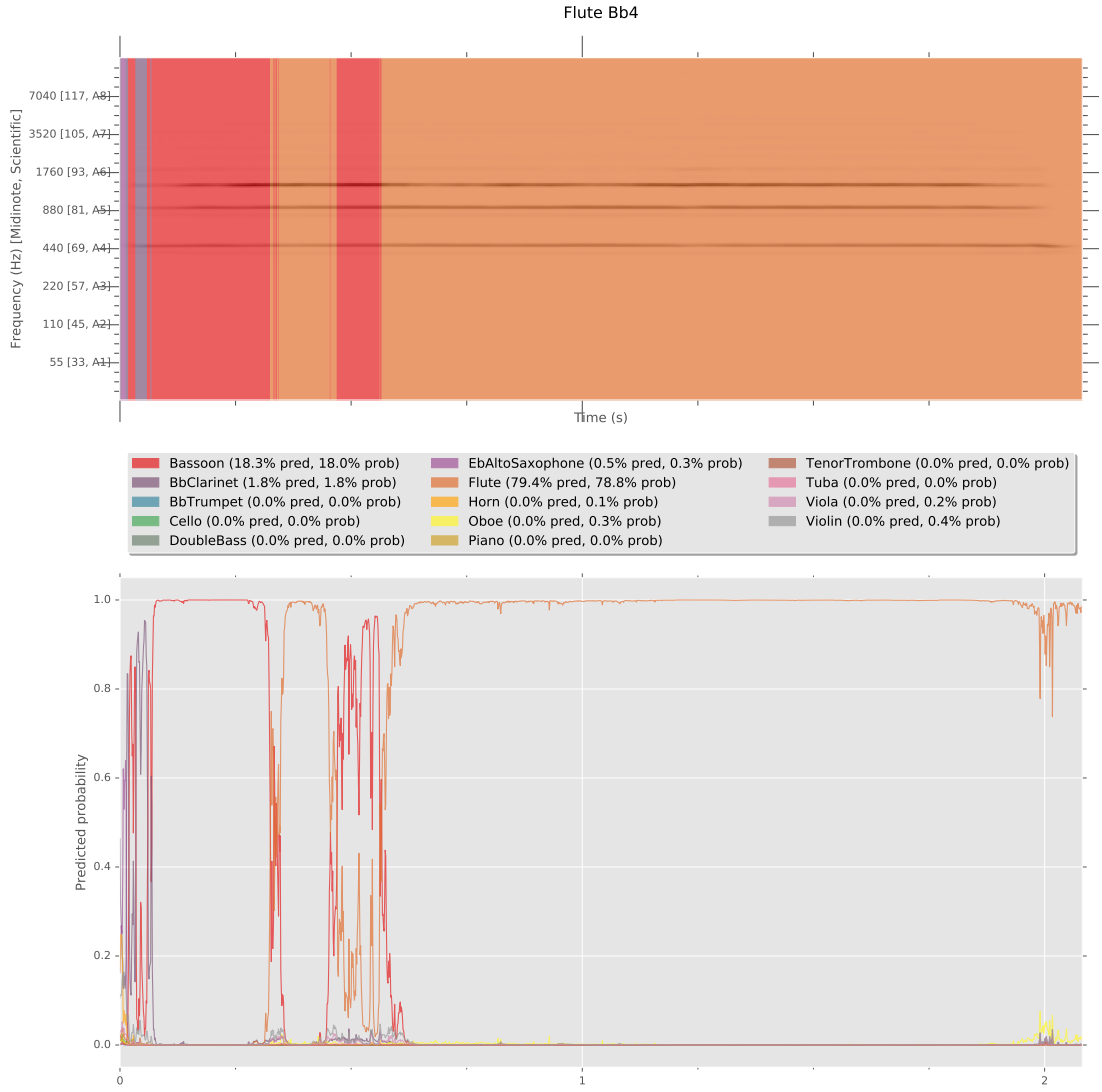


Figure 5.13: **CNN3 (trained on augmented data) note prediction**: An example frame-by-frame classification of an evaluation note by CNN3. The top plot shows the spectrogram of the note, overlaid with a colour representing the predicted class for each frame. The bottom plot shows the classifier's predicted probability for every class. The classifier performs similarly to the CNN3 trained with no data augmentation (shown in Figure 5.12), except it has an additional region of confusion.

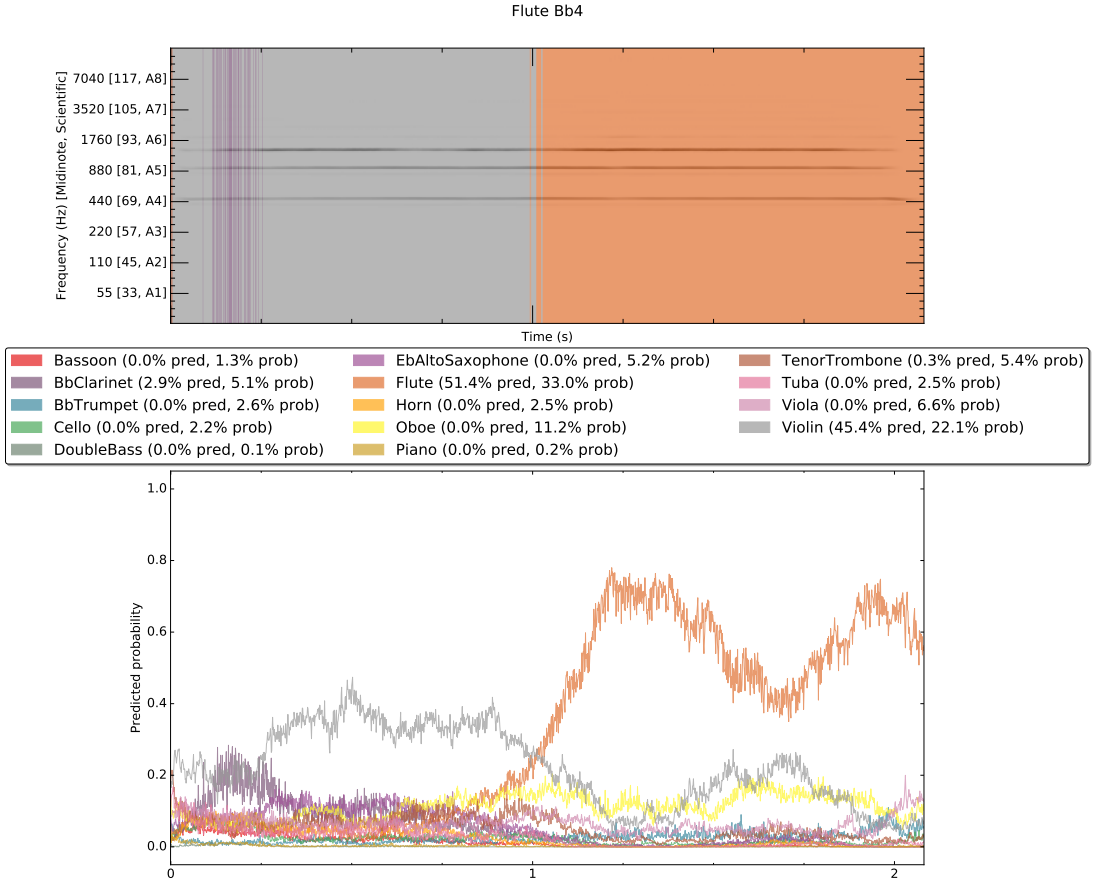


Figure 5.14: **Random forest (*not* trained on augmented data) note prediction**: An example frame-by-frame classification of an evaluation note. The top plot shows the spectrogram of the note, overlaid with a colour representing the predicted class for each frame. The bottom plot shows the classifier's predicted probability for every class. It is interesting to contrast this prediction with CNN3 (shown in Figure 5.12), which mistakes the onset for Bassoon.

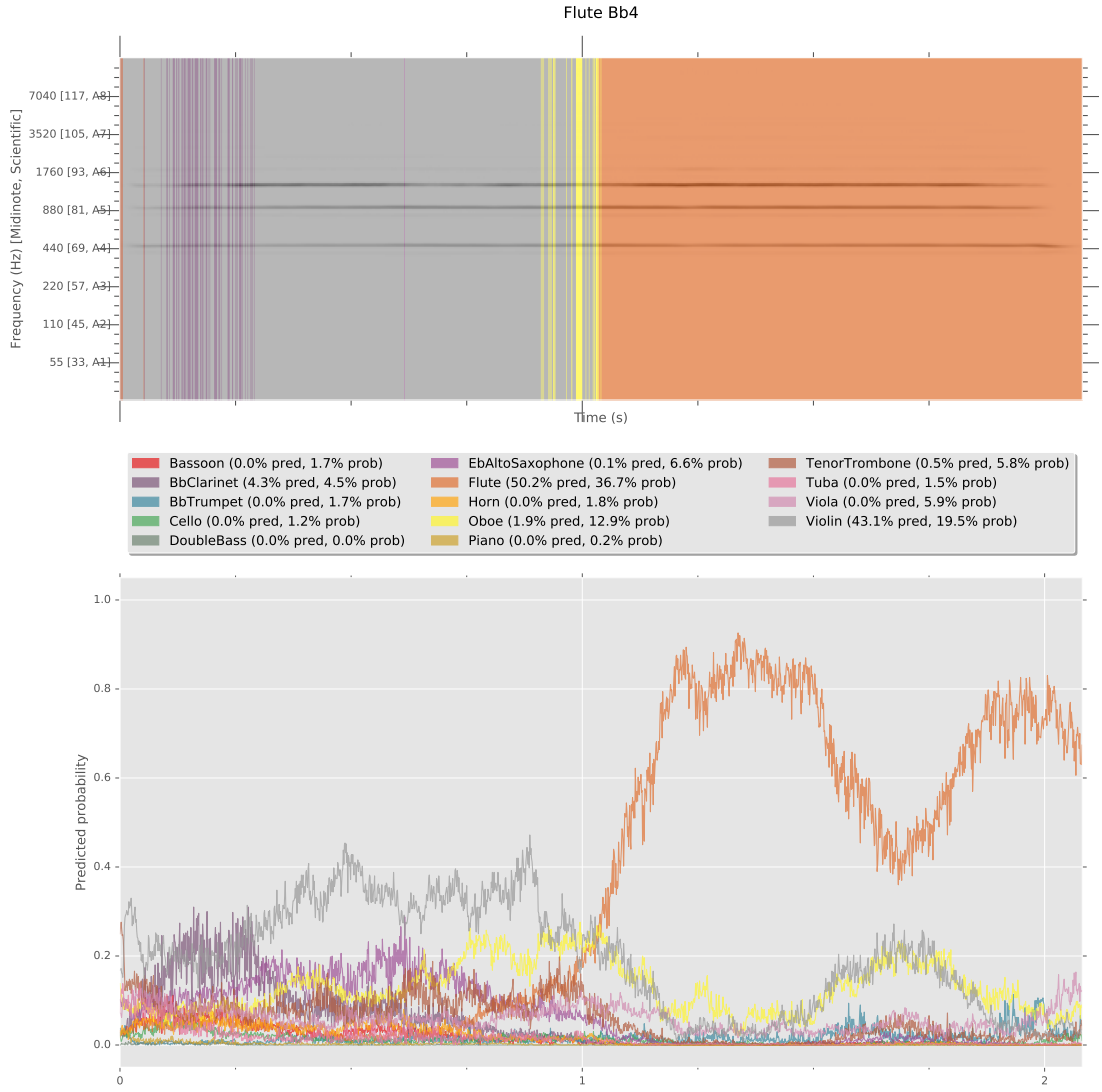
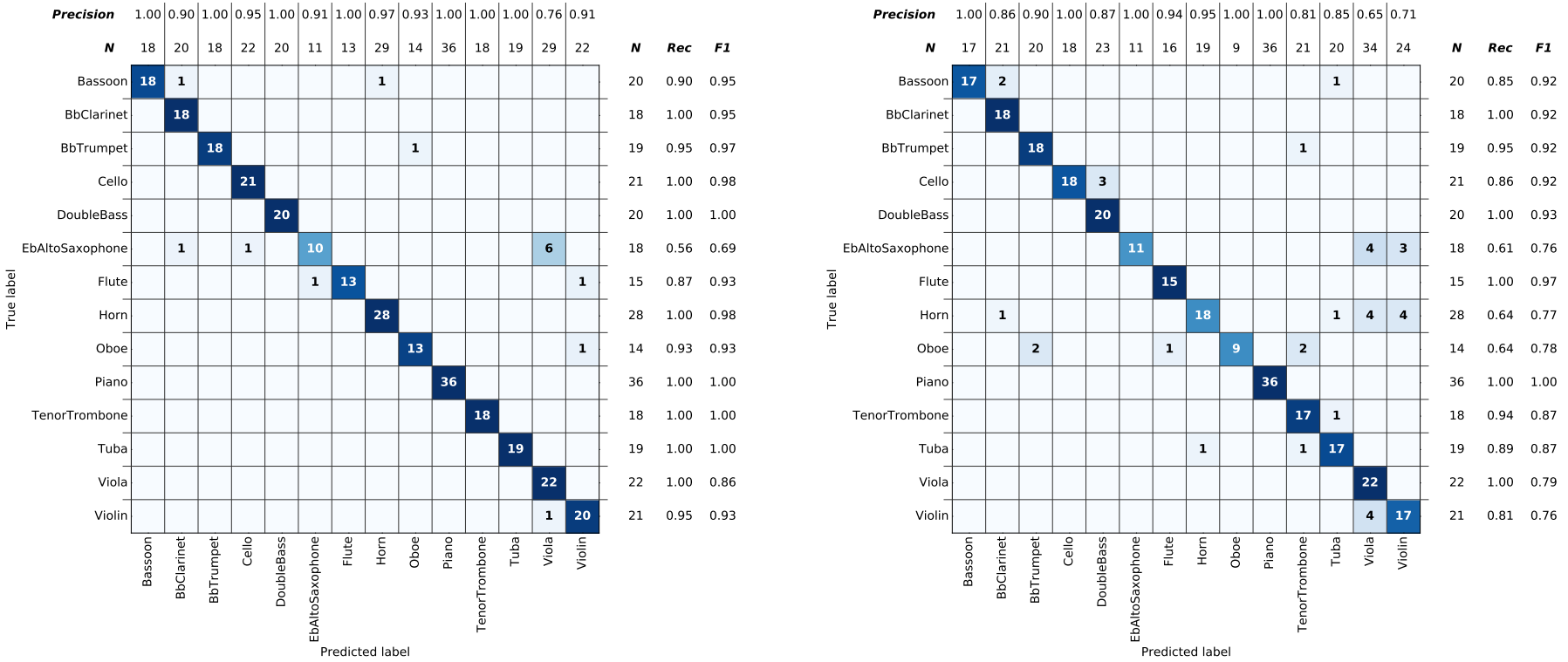


Figure 5.15: **Random forest (trained on augmented data) note prediction**: An example frame-by-frame classification of an evaluation note. The top plot shows the spectrogram of the note, overlaid with a colour representing the predicted class for each frame. The bottom plot shows the classifier's predicted probability for every class. As in the case of CNN3, the data augmentation hinders the prediction for this note, but by not nearly as much: only a few additional frames are misclassified.



(a) CNN3

(b) Random forest

Figure 5.16: **Note level confusion matrices**: Figure (a) shows the confusion matrix for CNN3 with the best training configuration, (b) the random forest. We can see that CNN3 performs well on all instruments except Saxophone, which is often confused for viola. Most notably, CNN3 is able discriminate between viola and violin, and is able to classify Oboe well. For a more detailed analysis, see the frame level confusion matrices in Figure 5.9.

Chapter 6

Conclusions

6.1 Headline findings

In this project, we have found that it is possible to train a system to perform instrument classification to super human levels of performance. Our main addition is that we find that this can be achieved by simply averaging the predictions for each column of a CQT. The CQT summarises signal information within a short temporal window for each time point; the size of the window depends on the frequency, and is designed such that there is no difference in temporal resolution. Whilst the longest windows (for the lowest frequency) are of a second in length, the shortest are of less than 0.001 seconds. Since our model treats all frames as independent, we now have a baseline with which we can compare models that try to model dependencies.

Additionally, our system is trained on 100 such frames from each note selected from less than half of each instrument's range, yet, in most cases, can generalise well to predict instrument notes it has never before seen. To our knowledge, this has not been explicitly demonstrated by state-of-the-art systems for instrument classification.

The relationship between the instrument, the pitch of a note, and the frequencies observed in a magnitude spectrum is not straightforward. We show that there is variation in the relative strength of the harmonics within single instruments, and also for the same pitch between instruments. We do not investigate variation between players, between different manufacturers of the same instrument, or different recording equipment – this could make an interesting future investigation.

6.2 Future work

6.2.1 Comparison with the state-of-the-art

In order that we can make any claims about the effectiveness of our CNN against the state of the art, we must take some further steps:

1. Generate MFCCs from the notes in the MIS dataset and compare our results with a classifier using these features
2. Compare with models by Tjoa *et al*. [47] and Grasis *et al*. [19] that do make use of temporal features
3. Re-run the experiment on multiple different datasets such that training data contains notes played by different musicians, recorded by different equipment, in different environments, and at different qualities

It may be of use to the community to create a standard dataset or download script for this task. We have compiled a list of common datasets for this task in Section 2.5.

6.2.2 Improvements to our current experiments

6.2.2.1 Preprocessing

We downsample our original wave data. Since we take a sample of frames from the spectrogram *after preprocessing* anyway, this was an oversight on our part. It most probably only results in information loss.

We opted for 'perfect rasterization, i.e. creating CQT spectrograms at a very high frame rate. The rate is two orders of magnitude higher than sigtia 2016 or humphrey 2011. There is probably no need for this level of detail. Indeed, an alternative fix to avoid bais towards lower frequencies would be to do a Variable Q Transform: increasing Q slightly with frequency. Brown recommends this approach in their original paper [7].

Finally, performing a stratified sample to create the data splits is recommended. We found that our less accurate performance on Saxophone could be due to a lack of training data for that instrument resulting from creating the data splits by randomly sampling the instrument-note pairs.

6.2.2.2 Modelling

Firstly we present improvements we could make to the CNN specifically, then move on to the describe higher level improvements to the pipeline.

Making predictions by sampling from our CNN could reduce bias and would not require any changes to the modelling process. Because the network was trained using dropout, instead of predicting using all the nodes, we could predict by randomly sampling nodes multiple times, to produce a mean predicted probabilities with uncertainty. Gal and Ghahramani posit that this is akin to casting a neural network as a Bayesian approximation in [16].

Ba *et al*. have very recently published an update to batch normalisation in [2], layer normalisation, which they claim to be a straight-swap improvement. This could be used in place of our batch normalisation process.

We haven't worked on incorporating prior knowledge about musical structure into our network training e.g. additivity. This should be introduced by way of weight constraints and, if performance remains the same, may give rise to more interpretable features.

Finally, the CNN convolutions are currently allowed to view the previous layers without focus. We could introduce striding, jumps between the focus of successive feature map nodes, to force net to look at different parts of spectrum implemented.

To make more general improvements, we could add a simple classifier on top of frame predictions (as opposed to taking mean of predicted probabilities), at the expense of including temporal information. We could additionally try other classifiers, such as the random forest extension 'extra trees' used in [1]. However, we applied the Scikit Learn model out-of-the-box, and found a big decrease in performance without, so some tuning may be required.

6.2.3 Further experiments

Our current experiments and results have led us to consider further work beyond the scope of this project. Our next focus will be to apply our method to polyphonic signals. For this, we intend to extend the work of Sigtia *et al*. in [43], and Huang *et al*. in [24], and employ powerful Recurrent Neural Networks with the latest additions to training techniques. There are other questions that have arisen along the way. Below we list a selection:

For instance, we observed that our model was unable to cope with a gap of half an octave in the Saxophone training data, even with data augmentation. Is this problem exclusive to Saxophone, or is it similar for other instruments? Is it possible to overcome this and design models able to bridge this gap? One method could involve the use of Adversarial Training [28]. Our idea is illustrated as follows. Imagine two people, Alice and Bob. You give Alice a vector $\boldsymbol{a}$, and Bob a vector $\boldsymbol{b}$. Alice and Bob have two objectives: to predict the instrument, and the f0 frequency from their respective vector. Your task is to generate an $\boldsymbol{a}$ and $\boldsymbol{b}$ such that: Alice can predict the Instrument, but is *totally unable* to predict the f0 frequency; similarly, Bob can predict the f0 frequency, but *not* the instrument. The result is that you will have trained a model to generate a representation for a note encapsulating independent qualities relating to the instrument ($\boldsymbol{a}$) and the f0 ($\boldsymbol{b}$).

Our experiments consider a limited set of instruments and playing styles. An obvious extension is extend this range, and explore whether the model handles non-tonal percussion instruments. For instance, it would be interesting to see if non-tonal could be classified as such without them being included in training. They are quite unlike tonal instruments; does our model of sound reflect this?

ISMIR 2016[1] has just published their proceedings. Specifically of interest to us are

[1] https://wp.nyu.edu/ismir2016/

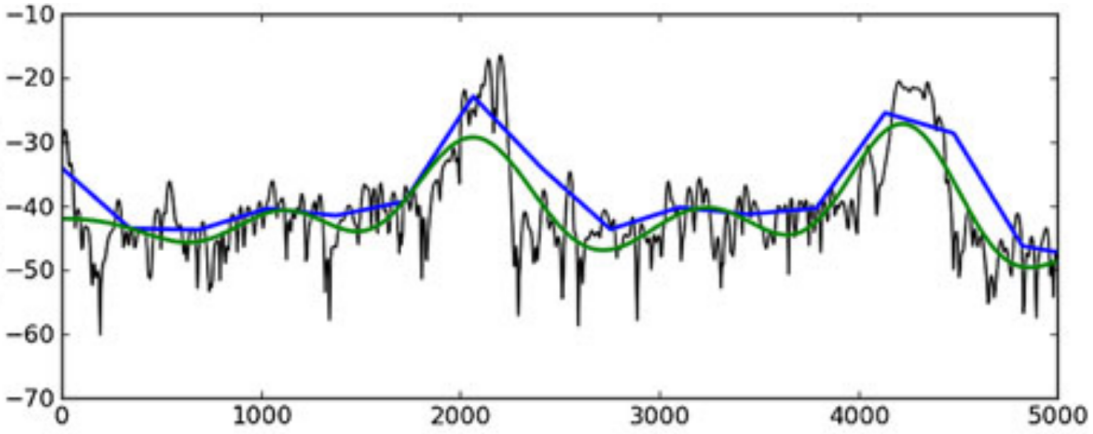


Figure 6.1: **The cost of spectral approximation**: Figure reproduced from [25] showing the log magnitude DFT coefficients (black) of a Violin signal against channel vocoder (blue), and MFCC (green) approximations. The MFCCs are a higher order approximation and thus able to better approximate the curve with the same number of coefficients.

the results of the MIREX Audio Tag Classification Competition[2], in which contestants apply tags, including some instruments, to 10 second audio clips. Can our methods be extended to this task?

More generally, it would be interesting to test the general theory that MFCCs are not resilient to noise, and observe whether the CQT, or our model extended to be applied to raw audio, is. This could be conducted by training a system on high, and testing it on low quality data (or vice versa). The issue of training on MFCC, or CQT abstractions is illustrated well by Humphrey *et al.* in [25], where they show how these features ignore the details contained in a DFT. We reproduce their illustration in Figure 6.1

Indeed, modelling directly from audio wave data is of interest to us. The we use the CQT as a low level abstraction, but it would be preferable to learn features from the lowest level. Humphrey *et al.* show that neural networks can represent the DFT in [25]. Actually learning this representation will present additional challenges. Is it even a good representation for the network to learn and use? We could test this by 'pre-training' networks, or simply initialising them with weights akin to transforms of interest, and reporting on performance for different tasks.

We trained a generative model as part of our experiments; namely a Naive Bayes model for classifying instruments trained upon relative frequency data. From this we observed a data distribution with characteristics reminiscent of statements from the literature e.g. woodwind instruments, such as the Clarinet have strong odd, and weak negative harmonics. This is shown in Figure 6.2. The performance to the model was similar to the logistic regression: pretty poor. Creating a more competitive generative model would be very useful to explain how the timbre of an instrument changes with the frequency of the note. It could also be used to synthesize interesting new instruments, and visualise the instrument space. Perhaps there are gaps within this space that could

[2] http://www.music-ir.org/mirex/wiki/2016:Audio_Tag_Classification

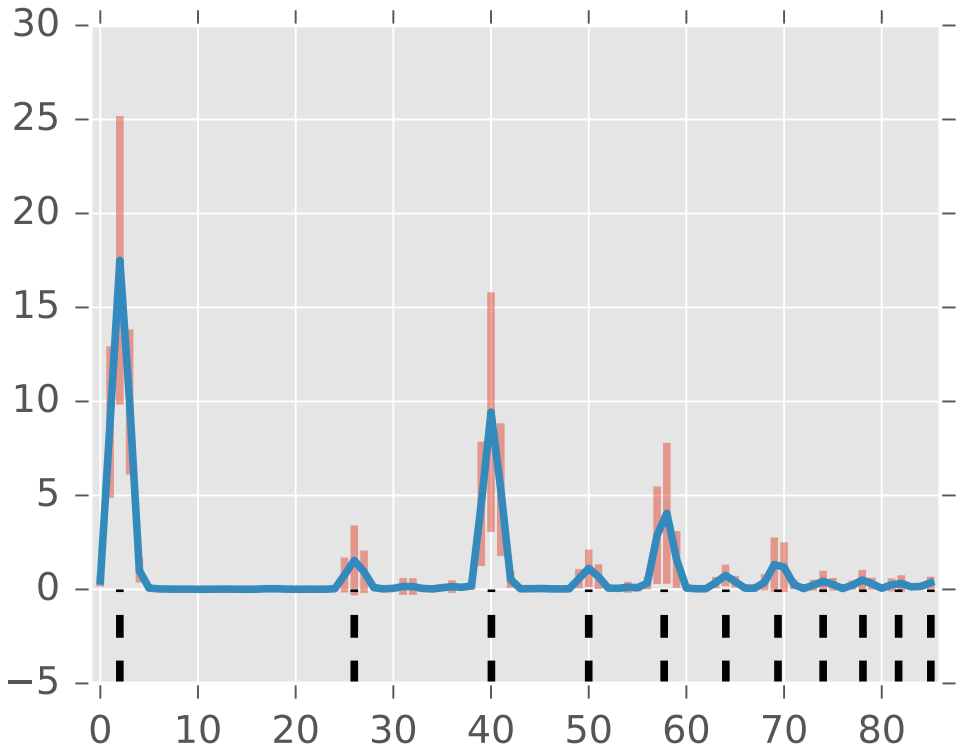


Figure 6.2: **The distribution of the relative harmonic space**: The B♭Clarinet note distribution resulting from the Gaussian Naive Bayes model in Experiment 1. The relative frequency is given on the x axis (with positions of harmonics shown in black). The distribution is plotted: the means are shown in blue, and the standard deviation in red. This distribution is expected from the literature, which states that odd harmonics are stronger for woodwind instruments [9], but the poor performance of the model overall suggests that the full story is much more complex

be filled by novel new synthetic instruments!

In [25], Humphrey *et al*. hint that the structure of musical signals provide challenges that neural network architectures for images may not be able to surmount without adaptation. We propose an adaptation to convolutional architecture: interval layers. An example of an interval layer could be a harmonic layer which would, like a typical convolutional layer, pass a filter over the spectrum but, unlike a typical convolutional layer, have a filter that looks at pixels harmonic distances apart. This addresses the non-local nature of harmonics. One potential advantage of this layer could be to reduce the number of parameters. We could also allow for the use of a DFT by encoding variable spacing into the layer parameters.

6.3 Closing remarks

PhD

Appendix A

Downloading the MIS dataset

The code contained in Listing A.1 will download all the data contained in the Iowa MIS database if the environment variable `MUSIC_PROJECT_HOME` is set.

```
#!/usr/bin/env bash
cd ${MUSIC_PROJECT_HOME}/data/input
wget -r -l5 --no-parent -A.aif http://theremin.music.uiowa.edu/
wget -r -l5 --no-parent -A.aiff http://theremin.music.uiowa.edu/
```

Listing A.1: Code to download the Iowa MIS database

Appendix B

Additional datasets from the literature

In addition to those described in Section 2.5, we found many datasets used for applications outside instrument classification but within music informatics. We list them here and hope they will be of use to the reader.

Table B.1: **Datasets use in other areas of Music Informatics other than Instrument Classification**: For datasets relating to instrument classification, see Section 2.5

Name	Description
MASS	12 short (˜20s) mixtures with sources for rock, pop, hip-hop, metal, bossanova, and reggae, with and without vocals, in WAV and MP3 format
BSS_Oracle	20 short (˜10s) 3-source tracks (mixtures encoded in matlab code), 10 entirely vocal readings, 10 musical
SiSEC2011	About 20 short (˜10s) mixtures with sources, half speech half music
dreanss_v1	Text file annotations for the drum parts within bss_oracle, mass, and sisec datasets
iKala	Wav files for 252 30 second excerpts of pop songs with lyrics complete with transcribed words and pitches
MAPS	A piano database for multipitch estimation and automatic transcription of music. 31 GB of CD-quality recordings in .wav format. Nine settings of different pianos and recording conditions were used. isolated notes and monophonic sounds, random chords, usual chords, pieces of music. The ground truth is provided for all sounds, in MIDI and text formats. The audio was generated from the ground truth in order to ensure the accuracy of the annotation.
WJazzD	MIDI information on thousands of Jazz solos on multiple instruments with chordal information. Contains links to the original recordings but not the recordings themselves. See the `record_info` table.

Name	Description
IRMAS	A solo instrument playing with accompaniment for 3 seconds from more than 2000 distinct recordings. 6705 audio files in 16 bit stereo wav format sampled at 44.1kHz. The instruments considered are: cello, clarinet, flute, acoustic guitar, electric guitar, organ, piano, saxophone, trumpet, violin, and human singing voice.
MIS	Solo unaccompanied instruments. Aiff recordings edited into chromatic scales played note-by-note at pp, mf, and ff dynamic levels throughout the range of the instrument. Some instruments were played with more than one technique, including arco, pizzicato, vibrato, and non-vibrato.

Appendix C

Colophon

This document was set in the Times Roman typeface using LaTeX and BibTeX, composed with TeXstudio on Linux Mint 17.2 Rafaela. The bibliography was generated using Mendeley [21].

Bibliography

[1] Jakob Abeßer and Christof Weiß. Automatic Recognition of Instrument Families in Polyphonic Recordings of Classical Music. 2015. [Cited in sections 2.1.1 and 6.2.2.2.]

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer Normalization. 2016. [Cited in sections 3.3.5 and 6.2.2.2.]

[3] Giuseppe Bandiera, Oriol Romani Picas, and Hiroshi Tokuda. good-sounds.org : a Framework To Explore Goodness in Instrumental Sounds. *Proc. 17th International Society for Music Information Retrieval Conference*, pages 414–419, 2016. [Cited in section 2.5.]

[4] Emmanouil Benetos and Simon Dixon. A Shift-Invariant Latent Variable Model for Automatic Music Transcription. *Computer Music Journal*, 36(364):81–94, 2012. [Cited in section 2.1.2.]

[5] Emmanouil Benetos, Simon Dixon, Dimitrios Giannoulis, Holger Kirchhoff, and Anssi Klapuri. Automatic music transcription: Challenges and future directions. *Journal of Intelligent Information Systems*, 41(3):407–434, 2013. [Cited in section 2.1.2.]

[6] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. Classification and Regression Trees, 1999. [Cited in section 3.4.4.]

[7] Judith C. Brown. Calculation of a constant Q spectral transform. *The Journal of the Acoustical Society of America*, 89(January 1991):425, 1991. [Cited in sections 3.1.4, 3, and 6.2.2.1.]

[8] Judith C. Brown. An efficient algorithm for the calculation of a constant Q transform. *The Journal of the Acoustical Society of America*, 92(5):2698, 1992. [Cited in section 3.1.4.]

[9] Nicolas D Chétry. *Computer models for musical instrument identification*. PhD thesis, 2006. [Cited in sections 2.1.1, 2.1, 4.1.2.3, and 6.2.]

[10] Antonio Criminisi and Jamie Shotton. *Decision forests for computer vision and medical image analysis*. Springer Science & Business Media, 2013. [Cited in section 3.4.4.1.]

[11] Sander Dieleman. *Learning Feature Hierarchies for Musical Audio Signals*. PhD thesis, 2016. [Cited in sections 2.2, 2.2, and 4.5.]

[12] Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, Jef-

frey De Fauw, Michael Heilman, Diogo149, Brian McFee, Hendrik Weideman, Takacsg84, Peterderivaz, Jon, Instagibbs, Dr. Kashif Rasul, CongLiu, Britefury, and Jonas Degrave. Lasagne: First release., aug 2015. [Cited in section 4.3.1.]

[13] Tuomas Eerola and Rafael Ferrer. Instrument Library (MUMS) Revised, 2008. [Cited in section 2.5.]

[14] Antti Eronen. *Signal processing methods for audio classification and music content analysis*. 2009. [Cited in sections 2.1.1, 2.1, 2.3.1, and 3.1.3.]

[15] Lawrence Fritts. "Musical Instrument Samples," Univ. Iowa Electronic Music Studios, [Online]. Available: http://theremin.music.uiowa.edu/MIS.html [Accessed 07/Aug/2016], 1997. [Cited in sections 2.5 and 4.1.]

[16] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation : Representing Model Uncertainty in Deep Learning. *Icml*, pages 1–10, 2015. [Cited in sections 3.3.4 and 6.2.2.2.]

[17] Dimitrios Giannoulis, Emmanouil Benetos, Anssi Klapuri, and Mark D. Plumbley. Improving instrument recognition in polyphonic music through system integration. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pages 5222–5226, 2014. [Cited in sections 2.1, 2.1.1, and 2.1.2.]

[18] Masataka Goto. Development of the RWC music database. *Proceedings of the 18th International Congress on Acoustics (ICA 2004)*, (April):553–556, 2004. [Cited in section 2.5.]

[19] Mikus Grasis, Jakob Abeßer, Christian Dittmar, and Hanna Lukashevich. A multiple-expert framework for instrument recognition. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8905:619–634, 2014. [Cited in sections 2.1.1 and 2.]

[20] Philippe Hamel, Yoshua Bengio, and Douglas Eck. Building musically-relevant audio features through multiple timescale representations. *Proceedings of the 13th International Society for Musical Information Retrieval Conference*, (Ismir):553–558, 2012. [Cited in section 2.2.]

[21] Victor Henning and Jan Reichelt. Mendeley - A Last.fm for research? *Proceedings - 4th IEEE International Conference on eScience, eScience 2008*, pages 327–328, 2008. [Cited in section C.]

[22] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, and Brian Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition. *Ieee Signal Processing Magazine*, (November):82–97, 2012. [Cited in section 2.2.]

[23] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv e-prints*, pages 1–18, 2012. [Cited in section 3.3.4.]

[24] Po-sen Huang, Minje Kim, and Mark Hasegawa-johnson. Joint Optimization of Masks and Deep Recurrent Neural Networks for Monaural Source Separation. *Taslp*, 20(1):1–11, 2015. [Cited in sections 2.1.3, 3.1.3, and 6.2.3.]

[25] Eric J. Humphrey, Juan P. Bello, and Yann Lecun. Feature learning and deep architectures: New directions for music informatics. *Journal of Intelligent Information Systems*, 41(3):461–481, 2013. [Cited in sections 2.2, 6.1, 6.2.3, 6.2.3, and 6.2.3.]

[26] Eric J. Humphrey, Aron P. Glennon, and Juan Pablo Bello. Non-linear semantic embedding for organizing large instrument sample libraries. *Proceedings - 10th International Conference on Machine Learning and Applications, ICMLA 2011*, 2:142–147, 2011. [Cited in sections 2.1.1, 2.2, and 4.1.2.1.]

[27] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167*, pages 1–11, 2015. [Cited in section 3.3.5.]

[28] Corey Kereliuk, Bob L. Sturm, and Jan Larsen. Deep Learning and Music Adversaries. *IEEE Transactions on Multimedia*, 17(11):2059–2071, 2015. [Cited in section 6.2.3.]

[29] Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, pages 1–13, 2015. [Cited in section 3.3.2.7.]

[30] Paul Mermelstein. Distance measures for speech recognition, psychological and instrumental, 1976. [Cited in section 3.1.3.]

[31] Univ. Pompeu Fabra Music Technology Group. "Freesound Project" [Online]. Available: http://www.freesound. org [Accessed 07/Aug/2016]. [Cited in section 2.5.]

[32] Michael A Nielsen. "Neural Networks and Deep Learning", Determination Press, 2015. [Cited in section 3.3.2.1.]

[33] Daniel Nouri. "nolearn: Abstractions around neural net libraries, most notably Lasagne." [Online]. Available: https://github.com/dnouri/nolearn [Accessed 07/Aug/2016], 2016. [Cited in section 4.3.1.]

[34] F Pedregosa, G Varoquaux, A Gramfort, V Michel, B Thirion, O Grisel, M Blondel, P Prettenhofer, R Weiss, V Dubourg, J Vanderplas, A Passos, D Cournapeau, M Brucher, M Perrot, and E Duchesnay. Scikit-learn: Machine Learning in {P}ython. *Journal of Machine Learning Research*, 12:2825–2830, 2011. [Cited in section 4.3.1.]

[35] One Laptop per Child. "OLPC Free Sound Samples" [Online]. Available: http://wiki.laptop.org/go/Sound samples [Accessed 07/Aug/2016]. [Cited in section 2.5.]

[36] H. H. Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175–184, 1960. [Cited in section 3.3.2.4.]

[37] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*. 1986. [Cited in section 3.2.2.]

[38] Jan Schlüter and Thomas Grill. Exploring data augmentation for improved singing voice detection with neural networks. *ISMIR*, 2015. [Cited in section 4.5.2.]

[39] Christian Schoerkhuber, Anssi Klapuri, and Alois Sontacchi. Audio pitch shifting using the constant-Q transform. *AES: Journal of the Audio Engineering Society*, 61(7-8):562–572, 2013. [Cited in section 3.1.4.]

[40] Christian Schörkhuber and Anssi Klapuri. Constant-Q transform toolbox for music processing. *7th Sound and Music Computing Conference*, (JANUARY):3–64, 2010. [Cited in sections 3.1.4 and 4.1.2.1.]

[41] Christian Schörkhuber, Anssi Klapuri, Nicki Holighaus, and Dörfler Monika. A Matlab Toolbox for Efficient Perfect Reconstruction Time-Frequency Transforms with Log-Frequency Resolution. *Proc. 53rd AES Conference on Semantic Audio*, pages 1–8, 2014. [Cited in sections 3.1.4 and 4.1.2.1.]

[42] C E Shannon. Communication in the Presence of Noise. *PROCEEDINGS OF THE IRE*, 37(2):10–21, 1949. [Cited in section 3.1.2.]

[43] Siddharth Sigtia, Emmanouil Benetos, and Simon Dixon. An End-to-End Neural Network for Polyphonic Music Transcription. *IEEE/ACMTRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*, 24(5):1–13, 2016. [Cited in sections 2.1.2, 3.3.2.6, 4.1.2.1, and 6.2.3.]

[44] Asha Srinivasan, David Sullivan, and Ichiro Fujinaga. Recognition of Isolated Instrument Tones By Conservatory Students. *Proc. International Conference on Music Perception and Cognition.*, pages 17–21, 2002. [Cited in section 2.1.1.]

[45] Hiroko Terasawa, Malcolm Slaney, and Jonathan Berger. The thirteen colors of timbre. *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 323–326, 2005. [Cited in sections 2.1, 3.1.3, 4, and 3.1.3.]

[46] Theano Development Team. Theano: A {Python} framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.0, may 2016. [Cited in section 4.3.1.]

[47] Steven K. Tjoa and K.J. Ray Liu. Musical Instrument Recognition using Biologically Inspired Filtering of Temporal Dictionary Atoms. *Proc. of 11th ISMIR Conf.*, (Ismir):435–440, 2010. [Cited in sections 2.1, 2.1.1, 3.1.3, 4.1.1, 5.1, 5.4.1, and 2.]

[48] Karen Ullrich, Jan Schlüter, and Thomas Grill. Boundary Detection in Music Structure Analysis Using Convolutional Neural Networks. *Proc. of the 15th International Society for Music Information Retrieval Conference*, pages 417–422, 2014. [Cited in section 3.1.3.]

[49] Gino Angelo Velasco, Nicki Holighaus, Monika Dörfler, and Thomas Grill. CONSTRUCTING AN INVERTIBLE CONSTANT-Q TRANSFORM WITH NONSTATIONARY GABOR FRAMES. *Artificial Intelligence*, pages 93–99, 2011. [Cited in sections 3.1.4 and 4.1.2.1.]

[50] Matthew D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv*, page 6, 2012. [Cited in section 3.3.2.6.]

[51] Fang Zheng, Guoliang Zhang, and Zhanjiang Song. Comparison of different implementations of MFCC. *Journal of Computer Science and Technology*, 16(6):582–589, 2001. [Cited in section 3.1.3.]