Deep Learning Guitar Tunings

One Semester Individual Project

Final Report

Author: Deeon Roy

Supervisor: Prof. David Marshall

School of Computer Science & Informatics

Cardiff University

May 2019

Abstract

Guitars can be tuned in a variety of tunings and take a master's ear to be able to detect this just from listening to the audio. In this project I aim to investigate possible machine learning (ML) and deep learning techniques to detect guitar tunings from audio. This audio will not be clean audio, i.e. there will be other instruments playing alongside the guitar along with other noise. This project presents a robust solution, comparing what combination of audio transformations and ML techniques work best in efficiency and accuracy.

Acknowledgements

Prof. David Marshall for excellent teaching in the field of DSP and the great enthusiasm he brings to subject to motivate me to complete this paper.

Chapter 1

Introduction

Strings resonate differently depending on frequency of the note plucked and string thickness, therefore having different harmonics when doing so. This is why, even when same note is played on a thick string, you can still hear the difference between that note being played on a thinner string. When tuning a guitar, the tension of the string is adjusted effecting the young's modulus of the string and therefore the frequencies it produces when plucked at different frets. Open strings will have a particularly will have a distinctively different sound and harmonics associated with them. An assumption made in this thesis is that all the information necessary to classify tunings are audible, and therefore available in the time-frequency domain. For the purposes of this thesis I will be using songs that contain primarily guitar and voice with little accompaniment.

The project can be broken down into these steps:

- Guitar audio gathered and tagged appropriately
- Split the audio into equally length segments to gain a large amount of sample data
- Use a variety of transformations into the spectral domain to test which ones work best with what algorithm.
- Feed the samples spectrogram data into machine learning algorithms to classify the tuning.
- Analyse data on how well the algorithm's trained and adjust.

Aim

The aim of this project is to develop a machine learning approach that can predict guitar tunings on a given song. To prove the system works, the model will be tested on unseen audio data taken from other artists and free samples.

Outcomes

The project presents an efficient and robust machine learning architecture for exact classification of guitar tunings, that can be used in future for more advanced audio transcription.
It also contains the development of a reusable toolset that performs pre-processing tasks and allows easy adjustment of network parameters alongside comparison and testing tools.
The focus is to have it be robust for acoustic guitars with minimal accompaniment however with more time and resources the system could be made to work for all types of guitar with larger amounts of noise.

Chapter 2

Background

2.1 Context

This chapter contains some background information necessary to understand for the context of the paper.

The Fourier Transform

The Fourier transform converts an audio signal that's in the temporal domain, where the signal is a function of time, and converts it to the frequency domain. This break down of a signal into its fundamental frequency components allows for spectral analysis and frequency manipulation. This transformation is integral to analysis of audio data as it allows us to visualise a signal in the form of a spectrogram.

In the context of this project we use the Short-Term Fourier Transform to plot frequency power against time.

Short Term Fourier Transform (STFT)

STFT is a lossless function so we don't lose any relevant information and works by applying a sliding window that overlaps applies a 1D Fourier Transform over the frequency bins for each window. The STFT is a very basic transformation that is fast and not very intensive. We compare this method with other transformations into the frequency space throughout the project.

Constant-Q Transform

The constant-q transform is related to the Fourier transform and is based off the STFT. It is a transform designed for musical analysis as it gives higher detail for a large frequency range. Due to the STFTs linear scaling factor, it compresses the lower frequencies into a small space and means they're less defined. Constant-Q uses logarithmically scaled frequency bins[1] and ensures the lower frequencies are well defined. This is important for detecting guitar tunings as noticing small differences in the way strings resonate is easier for the thicker strings. What determines the size of the bins is the Q factor, where:

$$Q = \text{centre frequency}/\text{bandwidth}$$

Cepstrum Analysis

The Cepstrum of a signal is the result of taking the inverse Fourier Transform of the logarithm of the expected spectrum of a signal. It's commonly used in the analysis of human speech. A Cepstrum contains information about the rate of change in different spectrum bands.

Discrete Cosine Transform (DCT)

The DCT is a lossy transform often used in image/video compression as its far simpler than FFT(Fast Fourier Transform). It works by converting a 2D matrix of

data into functions that represent the most frequent components in the matrix. By taking a subset of this data you can usually get the most important parts of data and reduce the amount of redundant data.

Mel-Frequency Cepstrum Coefficients (MFCC)

MFCCs main difference with a normal Cepstrum is the use of a linear Mel-scale that it tuned to the hearing of humans. MFCCs are calculated by performing STFT and then mapping the spectrum to the Mel-scale. Taking logs at the mel frequencies and applying a DCT on the Mel log powers gives us a spectrum where the amplitudes are the MFCCs. These MFCCs are then used as audio features.

This is commonly used in speech recognition and analysis due to its lossy nature, reducing computation time while still keeping the most important features of audio.

MFCC will be used as an alternative benchmark to Constant-Q and STFT to see if it's feasible to cut down computation time and keep high accuracy with a ML model.

Machine Learning

Machine learning is a term used to describe several algorithms that share the same principle. Using training data to discover relationships between data sets that can then be applied to unseen data to make accurate predictions. Supervised ML algorithms are those where you have labelled training data to train the algorithms and is what will be used in this project. There are two main types of supervised ML problems: Classification and Regression. Classification is where the output value of the algorithm is a strict category, e.g. "red", "blue". Regression is where the output is a real value such as "pounds", "weight".

This projects problem is a multivariate classification problem, where the categories are the guitar tunings, e.g. "EADGBE", "DADGBE".

K-Nearest Neighbour (kNN)

A kNN is a very simple ML algorithm, that is useful for its low calculation and training time. It's a non-parametric method that works by plotting data points on a plane. When classifying new data it will plot the point on the same plane and take the majority of its k nearest neighbours. K is the only adjustable parameter here and will directly effect the accuracy of the algorithm. Making k too large will cause the algorithm to pander to categories with the largest amount of points i.e. will classify the lesser used tunings as more commonly used tunings. However, making k too small will result in in-consistent results and may classify fringe data incorrectly.

[3]

Support Vector Machines (SVMs)

SVMs work by plotting each data item as a point on an n-dimensional space where n is the number of feature's present. The value of each feature then becomes a coordinate on this plane, these are the support vectors. Classification works by creating a hyper-plane that differentiates the classes the most. The hyper-plane that new data falls into determines what class it's classified as.

SVMs tuneable parameters include: Kernel, gamma and C. The kernels are used to convert a low-dimensional space into a high dimensional one. This allows us to convert non-separable problems into separable ones. Gamma is the coefficient used in the Kernel selected and a high gamma will try to make an exact fit on the training data resulting in overfitting. C effects the error term, this is the trade off between a smooth decision boundary and classifying points correctly.

Neural Networks (NNs)

NNs are a framework for ML algorithms based on biological neural networks found in animal brains. They are represented as a graph of neurons (nodes) that take an input and have a set of output nodes. Data is taken in and depending on the value passed through in the previous node a value will be passed onto a subsequent node until it reaches one of the output nodes.

Training an NN consists of adjusting the weights of each node in the graph. At the beginning of training, these weights will change drastically as it figures out the hidden relationships between data and their classification. As training progresses the weights will change less and less as it fine tunes the graph. Training is done over a series of epochs, where the same training set is passed through the network a number of times. The network adjusts it's weights based of this training set every epoch to reduce loss.

Measuring an NNs quality comes in the form of its loss function and accuracy. Where loss is the cost/error of a prediction, we are trying to reduce loss as much as possible, e.g. If a cat is 0 and a dog is 1, and we put an image of a cat into the network and it predicts 0.3, the loss is 0.3. Accuracy can be measured in a few different ways, in this project we will be looking at the Recall, Precision, F-1 and Accuracy scores to judge our network. Where Recall is the ability to find all correct predictions, Precision is the ability to not to label negatives that are positive, F-1 is an average of Precision and Recall and Accuracy is simply the number of correct samples predicted as a percentage.

Deep Neural Networks (DNNs)

Deep neural networks are a general term for NNs with hidden layers between the input and output layers. The stack of hidden layers has their weights adjusted to produce a reliable prediction. Deep NNs typically have a longer training time and have one main downside of having the hidden layers being a black-box. This means the interpretability of the algorithm is largely unknown and means we can't always see why it's making the decisions it is. However, after training, when a model is created, the model is generally very robust and performs in real time.

Convolutional Neural Networks (CNNs)

CNNs are commonly used for DNNs where the hidden layers are replaced by Convolutional layers and Max Pooling Layers. The Convolutional Layers work exactly like they do in Computer Vision, where they reduce an image input into its most important features, these are then followed by the Max Pooling Layers. These Max Pooling layers work by taking the max of the reduced image matrix allowing us to obtain the important features of a more abstract data set and therefore reducing over-fitting. The final layer then uses this data to make a prediction.

2.2 Tools and Software Used

Python

Python was the programming language used because of the number of packages available for Machine Learning and Visualisation. The ability to quickly test different techniques and visualise their data with ease was integral to completing a robust project in the time frame required.

Sci-Kit Learn (sklearn)

This python package is an all-round package used for machine learning. It contains many well-known algorithms with customisable parameters, along with tools for pre-processing, testing and visualisation in terms of any learning algorithm.

Modules used for pre-processing include: LabelEncoder, OneHotEncoder, StandardScaler, StratifiedShuffleSplit and GridSearchCV. These modules were used to allow automation of training and testing sets, labelling the data appropriately and preparing them for machine learning use. It was also used to create balanced training and testing data sets.

For testing and visualisation sklearn allowed easy creation of confusion matrices, classification reports and kept track of accuracy scores.

Sklearn also allowed us to quickly implement kNNs and SVMs with its "plug-and-play" modules and made saving our machine learning models and feature vectors for future use simple.

Keras

Keras is a high-level library for Deep Learning in python. It uses TensorFlow as backend, a framework for neural networks that is regarded as very future-proof. Keras is a commonly used library for deep learning for its extremely fast development time, while still giving accurate results and apt control over your network.
Due to Keras' high level nature you cannot look deep into array expressions and

optimise your network to up-most potential, like in PyTorch. However, due to its ease-of-use in implementing Deep learning models, it sufficiently reduced development time and allowed the experimentation of various methods in this project.

LibRosa

LibRosa holds invaluable tools for audio processing in python and is regarded as the default audio analysis library for python due to its large amount of features. It was used to create the various spectral representations of the audio data.

2.4 Practical Applications

This project has created a framework to be continued and used in further work in the field. A complete more robust product, accomplished with more time would be necessary for the following applications:

Audio Transcription

This work could be integrated into music transcription tools for professional and amateur artists allowing them to learn from music that has already been created. Currently transcription tools merely capture notes played but don't contain the tuning the guitar was in and often don't contain where the note was played on the guitar, which can heavily affect the sound and style of song. Due to the fact a note can be played on many places on the guitar, figuring out the tuning of the guitar before transcription is a necessary step to precise guitar transcription. The models developed here can be used to extend current applications already available.

Capturing Style of Contemporary Music

Research into capturing style of orchestral music has started to become popular and has had some success. However, due to the standard nature of how the instruments are played and tuned, fine details such as tunings of each instrument are not necessary. In contemporary music that uses guitars, different types/strings/tunings are used throughout and dictate the artists style and fingerprint. The product created here can be used in conjunction with other classification products for notation, and guitar type to build style profiles for music. Capturing style in this way has many applications including: Cultural preservation and remastering of old/damaged audio, Creative tool for artists allowing the model to fill in gaps in their music based off their style or to create new music based of artists that may have passed.

Interactive Media

There are many video games and interactive experiences that use music as the core element. Things like Rockband, Guitar Hero, BeatSaber etc. give you the task of imitating pre-recorded tracks for points. These tracks have to processed and programmed into the game by hand, meaning users can only play with tracks developers have decided to convert into the system. However, with the ability to detect tunings, further research into precise transcription could mean that you have a system that allows users to plug in their own music of choice into these interactive experiences.

Problems

For the applications stated above, the main problems arise from needing high bit rate annotated samples that don't contain much noise or polyphony. However, due to this product being able to detect tunings in noisy environments with relatively low-quality audio, it may mean that making precise predictions on transcription becomes substantially easier because of the new information that can be assumed. By obtaining tunings, using a combination of music theory and how strings resonate at different lengths, along with methods to separate signals to detect where notes were played without having clean source samples.

These applications are discussed in more detail in Chapter 6.1 Further Work.

Chapter 3

System Overview

3.1 Overview

The project attempts to prove that using machine learning methods is feasible for solving the proposed research question. The initial focus is therefore to show that the solution has potential, followed by research into how the solution can be optimised for better performance. In order to accomplish this a framework has been designed as a test bed to compare and experiment different approaches to solving the problem in a machine learning space. The framework employs a library of functions that have been designed to be modular and abstract away all non-essential details from the development framework. Resulting in a clean easy to understand system that streamlines development.

This project has a strong focus on what combinations of data representation and machine learning model combine to achieve the best results and is thus broken down into 3 main sections:

- **Data Collection**
- **Pre-Processing**
- **Network Architecture/Configurations**

The framework has been designed to iteratively implement all 3 tasks and is outline in Chapter 4: Implementation.

Project Structure

During development, the optimisation of networks hinged on constant evaluation and performance of current solutions to the problem. This meant that the initial focus of seeing if deep learning methods had potential was answered very quickly and improvements could be made aptly. The outline of development is represented in these steps:

1. Research: Researching NN architectures, Current solutions to similar problems, Data Representations, Pre-processing
2. Data Collection: Obtaining data, downloading and sorting it.

3. Framework Development: Create a reusable platform that has all the tools to test various network architectures, data representations that logs findings.
4. Experimentation: Perceiving performance of different network architectures with different data representations
5. Optimisation: Changing hyper parameters of networks and pre-processing elements.
6. Evaluation: Analysing differences in results and interpreting which changes allowed for the best solution.

Steps 1-3 were done at the beginning of the project and were not iterated over again. With steps 4-6 were iterated over many times to achieve a good result.

3.2 Data Collection

Data collection for this project was rather simple, it consisted of using Joni Mitchell songs supplied by Prof. David Marshall. The Joni Mitchell song set proved invaluable in this project, as she uses a large variety of standard and irregular tunings. The folk nature of her sound also lends itself well as it consists mainly of guitar and voice with few other instruments. This along with free guitar samples lent itself well to the timespan of the project as training data on more noisy environments could prove time consuming. Once the data was downloaded it needed to be marked with the correct tuning, converted to wav and sorted into its appropriate tunings. Marking the data appropriately consisted of prefixing the filenames with the appropriate tuning.

Due to the nature of DNNs, a large set of sample data is required to train. To artificially produce more sample data, audio files were split into small frames and overlapped with each other over a window. The system iterates over a directory containing appropriately named wav files and splits them aptly, prefixing the tuning name for labelling later on. Different configurations of audio frame and window size were experimented with in the project, to see if it had an impact.

3.3 Pre-Processing

Before the audio data is ready for use in the chosen networks, a large amount of pre-processing needs to be accomplished. How the data is processed before entering the network may heavily affect its performance and was experimented with to achieve the best results.

Data Representation

The audio data needs to be represented in some relevant format for the networks to find relationships between the waveforms and guitar tunings. The simplest way would be to sample amplitude at regular spaced intervals at a rate at least twice the highest frequency of the waveform (Sample Rate). On the grounds of Nyquit's Sampling theorem, any lower would lose perceptual quality. The most common sample rate being 44100Hz (CD Quality), which

captures good quality of audio without much redundant data. Data represented in such a way is described as an array of samples/frames.

Using an array of samples is the rawest form of data representation, however it is very computationally difficult to create any meaningful relationships between audio presented this way and the harmonics of the strings played. This is why the audio data needs to be converted into the time-frequency domain. Pitch and harmonics are defined by the frequency of a sound, by transforming the data into the frequency space, relevant semantic features of the signal can be emphasised. This requirement of the project enabled experimentation into which representation offers the best performance.

Here we experiment with 3 different time-frequency transformations to accomplish this:

- STFT – lossless, basic baseline
- Constant-Q – STFT with optimisations for musical analysis
- MFCC – lossy representation that captures audio data perceived by the human ear.

STFT serves as the baseline representation and works as a great benchmark to compare Constant-Q and MFCC implementations. MFCC has been included by virtue of its performance in similar audio-recognition tasks; particularly in voice-recognition. It performs particularly well in voice-recognition tasks as the Mel-Frequency log scale is tuned to human perception of audio. This generally means that some musical information about the signal maybe lost; however, guitar maestro's can often tell a guitar's tuning just through ear so it's possible that most of the relevant information for classification is still available. MFCC's lossy representation also leads to it having a shorter computation time.
All 3 of the representations have a set of parameters that can be tuned to emphasise the signal in different ways and will be experimented with.

3.4 Network Architecture and Configuration

The problem proposed is considered a multivariate classification problem, with the input shape being the array shape of the data representation e.g. Constant-Q's input dimensions will be frequency bins x time. This varies between STFT, Constant-Q and MFCC, so the input shape needs to be variable. The output shape is the number of classifiable tunings, in our case 4. Problems of this nature prove easy to evaluate as the accuracy of predictions is very clear cut.

Grid Search

The number of different networks being tested made it necessary to create an efficient and comprehensive system for hyper-parameter selection. Utilising a grid search allowed multiple parameter combinations to be tested without the need to manually change them. The grid search works by running all possible parameter combinations specified in a dictionary, performs cross validation and returns analytics on all the configurations tested. This was necessary to significantly reduce development time without hindering the integrity of the experiments.

Performance Metrics

For a supervised network to learn the correct weighing's of nodes, it requires accurate feedback on its performance. The feedback it receives is called the performance metric; this is what the network attempts to improve when adjusting its weights.
The chosen metric for this problem was Categorical Accuracy. This is very simply the number of correctly classified samples and is calculated at the end of each epoch contrary to other common metrics used. This metric was chosen on the grounds that the dataset contained some samples that did not contain any guitar audio. The amount of "false samples" in each category was unknown and therefore could affect results more drastically if another metric was used, such as one that would incorporate precision.

These sections of guitar absence could not easily be trimmed for silence as there was still voice and piano playing. To void these samples the data requires to be annotated with instruments and time stamps. This could be solved by using a musical instrument classifier, the proposed solution is discussed in Chapter 6: Further Work.

Sequential Deep Neural Network

There are 2 ways to build Keras models: Sequential and Funcitonal.

Sequential models be the simplest of deep learning models used here. They do not allow the user to define models that connect to more than just previous and next layers like functional networks allow for. However, they are simple to implement and run, often producing good results with a fast training time. In the current implementation, no processing is performed on the data as it passes through the sequential model, allowing it to train much quicker compared to the CNN counterpart.

Convolutional Neural Networks

Convolutional neural networks are already heavily used in computer vision applications and have seen a lot of success. They work excellently to reduce over-fitting by abstracting data and finding only the most relevant features within image data. Overfitting was reduced further with the inclusion of some dropout layers. This was to increase the generality of the network, due to the training data being from a small selection of songs by the same artist. Here we treat the spectral representations of the audio frames as the "images" and use a 2-D convolutional network.

Chapter 4

Implementation

4.1 Overview

A package called ml_gt has been created that includes methods for **preprocessing, learning** and **evaluation**. This library is then used to implement code used in experiments. A framework has also been designed

to streamline the experimental process.
All of the machine learning approaches attempted use the same feature vectors and therefore meant that a streamlined system could be created as feature vectors would not need to be calculated multiple times, but saved as files to be used later.
The target systems for this product are Windows and Linux systems with CUDA-capable GPUs. OSX is not targeted because its lack of Nvidia GPU utilisation, therefore taking substantially longer to train the networks.

4.2 Data Preparation

Input Files

The audio files used in this project did not have naming conventions relative to guitar tunings. This was problematic, since the labels used in classification were determined by the filename. This meant changing filenames accordingly and sorting audio files into appropriate directories. Artists were put in separate directories and files were named with their tuning as a prefix. E.g. “EADGBE – All I Want.wav”. To keep everything consistent, all files were also converted to wav files before being put through the system.

Splitting Audio Files

The function split_audio and split_folder found in ml_gt.preprocessing have been used to create audio slices from the original long form audio. To artificially increase the number of samples gained from a single song, overlapping the split audio files with a window was experimented with. To accomplish this scipy.io’s wavfile function was used to read and write wav files. Once the wav file was read in the function would merely iterate over the signal with a step size of (chunk size-window length) and write the wav files, splitting the different tunings into folders and naming the files aptly.

Labelling

A function named create_labels was created to save a list of labels that corresponded with the training data set so that all approaches could merely import labels without having to create them each time. Sklearn’s LabelEncoder function was used to encode labels into integer formats so that sklearn could utilise them in its functions. For the deep learning models, where keras was used, labels also needed to be one hot encoded and are done using sklearns OneHotEncoder function.

4.3 Sample Pre-Processing

Data Representation

Using the labelled data generated by functions outlined in Data Preparation, we need to now convert the raw waveforms into the time-frequency space as mentioned previously. These transformed representations act as the feature vectors used throughout the ML algorithms. Librosa’s STFT, CQT and MFCC functions were used here.

Short Term Fourier Transform

The STFT function in Librosa is implemented well with good default settings and is a good benchmark to work and compare off. STFT is a very standard representation and doesn't amplify any musical features making it a great starting point. If the network can create successful relationships between an STFT representation and the tuning, the more specified representation should generate much better results.

Constant-Q

Librosa's implementation of constant-q allows for many parameters to be changed, however the most impactful parameters that could be changed were, minimum frequency and number of frequency bins. Librosa's default min frequency is a C1, at 32.70~ Hz, however this was changed to an E, at 41.2~Hz (MIDI note 28). The reasoning is that on a standard guitar tuning, the lowest string is E at 82.4~Hz and the likelihood of this being tuned a whole octave down is unlikely. However, as the lowest string can often characterise the rest of the tuning, it's vital that detail in the lower frequencies aren't lost, hence the chosen minimum freq. The number of frequency bins was experimented between 37-88. The range decided was based on the fact that guitars up to the 12^{th} fret have 37 different notes whereas pianos, and what the western scale is based off have 88 different notes.

MFCC

The main parameter that is controllable here are the number of mel frequency banks and number of MFCCs. Due to the nature of MFC, the lower frequencies are given a higher resolution regardless so stayed with the default number of frequency banks (128). The number of MFCCs however, effect the dimensionality of our feature space and therefore the networks performance, so lower numbers like 13 and 20 were experimented with. MFCC's use of DCT to allows for lower numbers of coefficients as one of DCT's properties is that it de-correlates and keeps most of the information in the first few coefficients.

4.4 Parameter Selection

Tuning hyper-parameters can hugely affect the performance of any ML algorithm. Although not all possible configurations were able to be tested, most popular configs were tested.

GridSearchCV

Sklearns GridSearchCV allowed for a streamlined method for testing multiple permutations of parameters while displaying their performance and return what the parameters were best wrt. a performance metric.
The GridSearchCV() function takes a function of the model being used as the first parameter. This gives the ability to use custom models and meant that Keras models could be tested as well as the out-of-the-box sklearn networks (SVM and kNN).
The parameter ranges that were tested are as follows:

kNN

Parameter	Possible Values
Weights	**Uniform, Distance**
Neighbours	**1, 2, 5, 10, 15, 20**

SVM

Parameter	Possible Values
C	**Logspace(-1, 10) Steps = 13**
Gamma	**Logspace(-9, 3) Steps = 13**
Kernel	**rbf**

DNN – Sequential Model

Parameter	Possible Values
Layer Config	**[12], [12, 6, 6], [6 6 6], [6]**
Activation	**Relu, tanh**
Epochs	**100, 150**
Batch Size	**5**

CNN Model

Parameter	Possible Values
Layer Config	**[16, 16, 16], [16, 16], [16, 12], [12, 12], [12, 6], [6, 6]**
Activation	**Sigmoid**
Final Activation	**SoftMax**
Optimiser	**Sgd**
Epochs	**150, 200**
Batch Size	**50, 100**

Kernel rbf was decided on as the first kernel to try, due to its common use for classification tasks with non-linear datasets. It performed well on the first run through and therefore did not need to be experimented with.

Due to the high performance of the SVM and kNN classifiers, it was estimated that a large epoch and batch size would not be necessary and would take up unnecessary time. This is why they were kept to relatively low numbers.

For the activation function (not including the final layer), two popular non-linear activation functions were decided on, relu and tanh. Non-linear activations although are more difficult to train, achieve much better results as linear activation functions cannot learn complex mappings.
Tanh was decided on over sigmoid as they have very similar shortcomings, while tanh being easier to train. Relu is the de facto default for NNs that do

not use reinforcement learning and is relatively easy to compute with good results.

4.5 Neural Network Architecture

Training Parameters

CNN

Layer (Type)	Parameters
Input	**Input shape = Spectrogram Dimensions with added 1st and 4th layers**
2D Conv Layer	**Filters = 12, Kernel Size = 3, Activation = Sigmoid**
2D Max Pooling Layer	**Pool size = (2, 2)**
2D Conv Layer	**Filters = 12, Kernel Size = 3, Activation = Sigmoid**
2D Max Pooling Layer	**Pool size = (2, 2)**
Flatten	
Dense	**Units = 12, Activation = Sigmoid**
Dense	**Units = (no. of tunings), Activation = SoftMax**

Figure X: CNN Architecture

DNN

Layer (Type)	Parameters
Input	**Units = 12, Activation = relu, Input shape = Spectrogram Dimensions**
Dense	**Units = 12, Activation = relu**
Dense	**Units = (no. of tunings), Activation = SoftMax**

Sigmoid Layer

Sigmoid layers are used to normalise data, independently of each other. It will squash a vector into a range between (0,1), independent of each class. Sigmoid layers are often used as output layers for multi-label classification problems due to this. This is used to normalise the vector before applying the SoftMax layer.

SoftMax Layer

A SoftMax layer works very similarly to a Sigmoid layer in the way it converts a vector into a range between (0,1) however the sum of all values in the vector should equal to 1. A SoftMax layer was used for the output layer of the network as the problem is a multi-class classification problem where each input can only be 1 specific class. The sum of all vectors equalling 1 is useful here as when increasing the value of one output class the rest lower, highlighting one specific class, which may not happen in a Sigmoid layer.

Loss Function

Categorical Cross Entropy was decided on for the use of the loss function. Cross Entropy loss is the measure of performance of a classification model which has a probability between 0 and 1 as output. Cross entropy loss increases as the predicted probability diverges from the ground-truth value, i.e. the perfect model would have a loss of 0. Categorical Cross Entropy loss, also called SoftMax loss, work by applying a SoftMax calculation before applying Cross Entropy. Cross Entropy is calculated for each individual class then a gradient expression is used to calculate a loss score that considers negative class probabilities and positive class probabilities.

Dropout Layers

Dropout layers are a regularisation technique for NN models that are used to help reduce overfitting. They work by "dropping out" random neurons in the network when training, setting some of the input vectors values to 0. This has the effect of meaning no one neuron is being relied on to classify the tuning. It was key to include these dropout layers due to the similarity of the training set. Using primarily Joni Mitchell tracks means that the network has a high potential to become fragile and pick up on different features to recognise the tuning, such as style of play when using a specific tuning. This was potentially dangerous due to the fact most tunings were found in the same albums.
Setting the dropout layers probability to 0.4, a relatively large value, applying dropout to both convolutional layers seemed to suffice. This created a more robust network when tested on different artists and albums tracks.

4.5 Evaluation

A few logging features were created so that during and after training we could evaluate the systems performance. During training Keras' CSVLogger is used to identify areas where the networks accuracy and loss reach optimal levels, by logging loss and accuracy over each epoch. This allows use to recognise when certain configurations reach peak performance much earlier/later than others.
Alongside this txt files are created with parameter configurations for the spectral analysis, network and audio slice size. These txt files also include which files the network predicted wrong so that they can be inspected further. This was useful, as separation of the audio data was not implemented, meaning that the network may pick up audio with no guitar (just voice and piano for example) and identify it incorrectly. Inspecting the files that were classified incorrectly gives us a better idea of the networks performance.
Before and after training all necessary files are saved using pickle, numpy and keras to .pl. npy and .h5 files respectively. These include the model, encoded layers, feature vectors etc. so that a model can be initialised without the need for retraining. Saving all the other necessary data also allowed for training of different network configurations without the need to recalculate feature vectors.

4.6 Problems encountered

GPU Memory Allocation

During the parameter evaluation for the CNN when using the grid search, CUDA threw an unexpected memory allocation error. This had happened previously when first implementing the CNN but was solved by changing the tensorflow config to allow for growth, utilising all the GPU's available memory.
The memory allocation error occurred late on in development, while the network had trained for many hours overnight. Initially this was a huge setback, however the issue was resolved by training the network in batches instead of at 1 epoch at a time.

Chapter 5

Results

All experiments were conducted using the network parameters outlined in the **Implementation** section. Results were gathered using a windows 64-bit System using a GTX 1060 6GB GPU, i7 6700 and 8GB RAM. Performance was evaluated in a number of ways.

5.1 Grid Search Results

SVM

```
  warnings.warn(CV_WARNING, FutureWarning)
best params are {'gamma': 0.0001, 'kernel': 'rbf', 'C': 100.0} with score 0.99
Recall:  [0.97916667 0.99593496 0.99470899 0.97740113]
Precision:  [1.         0.97609562 0.98947368 0.99425287]
F1-Score:  [0.98947368 0.98591549 0.99208443 0.98575499]
Accuracy: 0.99  , 747
Number of samples: 756
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       144
           1       0.98      1.00      0.99       246
           2       0.99      0.99      0.99       189
           3       0.99      0.98      0.99       177

   micro avg       0.99      0.99      0.99       756
   macro avg       0.99      0.99      0.99       756
weighted avg       0.99      0.99      0.99       756
```

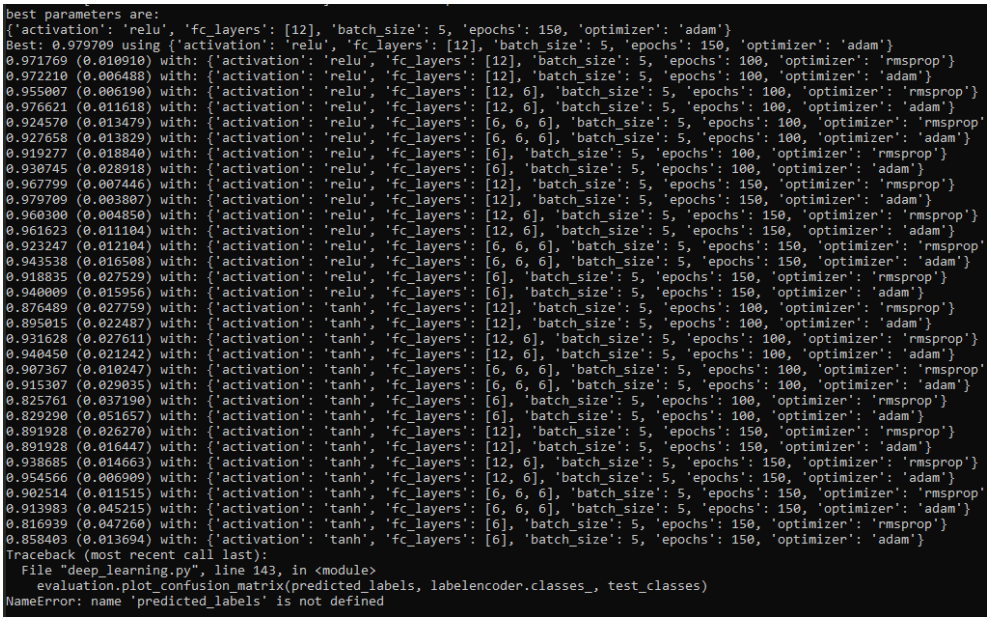
kNN

```
best params are {'n_neighbors': 1, 'weights': 'uniform'} with score 0.98
Recall:  [0.98611111 0.95934959 0.97894737 0.96590909]
Precision:  [0.95945946 0.97520661 0.95876289 0.98837209]
F1-Score:  [0.97260274 0.96721311 0.96875    0.97701149]
Accuracy: 0.97  , 367
Number of samples: 378
              precision    recall  f1-score   support

           0       0.96      0.99      0.97        72
           1       0.98      0.96      0.97       123
           2       0.96      0.98      0.97        95
           3       0.99      0.97      0.98        88

   micro avg       0.97      0.97      0.97       378
   macro avg       0.97      0.97      0.97       378
weighted avg       0.97      0.97      0.97       378
```

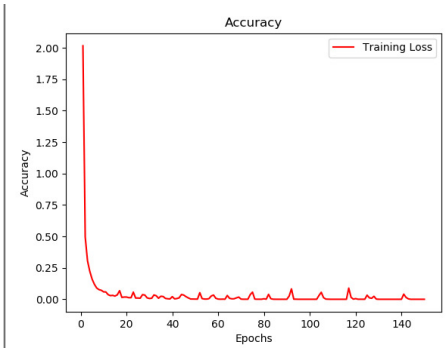
DNN



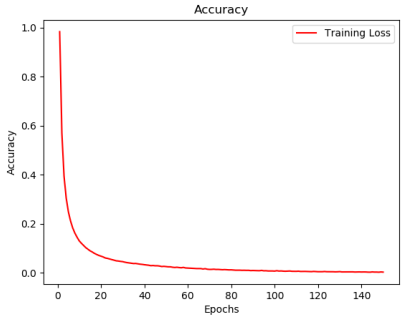
CNN

5.2 Summary of results

What was discovered through the results was that the spectral representation of the audio data had very little impact on the final performance of the networks. However, MFCC achieved peak performance the quickest in all networks tested, with no spikes in the loss graph. The MFCC representation also being the fastest to compute feature vectors for makes it the ideal candidate for tuning classification tasks.



CQT – DNN



CQT – MFCC

5.3 Evaluation of results

It has been determined that the neural network architecture effects the accuracy of a model much more than spectral representation.

Chapter 6

Conclusion

This project aimed to evaluate the feasibility of using machine learning techniques to detect guitar tunings from audio. It has been shown that is possible, with a variety of solutions being explored.
The average time for training machine learning models was around 15 minutes for the CNN, if cross validation was used 5 times and around 12 mins for the normal DNN. Peak performance was reached early on for the deep neural network models, especially the regular non-Convolutional model. This suggests with more performance tweaks and better sample choice, that the Network models could yield better performance, possibly training quicker and with fewer resources, classifying data outside of the original training space (Joni Mitchell tracks) with higher accuracy.
The ml_gt library created takes already existing solutions and combines them with modularity in mind. This provides a good abstraction for the tools used in this project and has been designed in this way to allow easy application into further work.
It is encouraging that simple off-the-shelf networks would yield such great results in this task, especially as it was not expected that the simpler machine learning techniques would perform so well. This is promising for the prospects of future work as what has been found here can be taken into more complex spaces with a strong base to build off.

Another approach to this project could have focused on designing an optimised network, with possible encoder/decoder integration. However, converting Keras code into pure TensorFlow or PyTorch proves a time-consuming task due to the high learning curve presented, especially due to the lack of documentation available compared to Keras. However, in consummation, it has been proven useful to display that there is negligible difference in the spectral representation used when the network has been trained on a relatively small and simple data set.

As a whole this project has been a great learning experience, with much unexpected success and many problems along the path. It has been completed with a great deal of scope left for future work and created in a way that should make it easy to adapt.

6.1 Further Work

Future work should first focus on the current problem and structure, optimising the system for larger problem spaces. As explained before, this work has also been designed to be used alongside current recognition and classification solutions to improve them or create something new entirely. Some possible avenues are discussed below:

6.1.1 Current Problem Structure

Optimise Processing Pipeline

Only the most popular network configurations were decided on for the networks used in this project due to time constraints on training the data, with the current hardware available. This has meant that possible more niche configurations and larger networks were shied away from, that may have been more optimal. Two possible solutions that could be used separately or together are:

- **Parallelise Time-Frequency Transformations:**
  There has been some work on performing FFT on multiple cores in parallel for python[7]. This could be adapted to also perform specific transformations such as MFC and CQT. This would allow for multiple feature vectors to be created in a much shorter time.
- **Asynchronous Pre-Processing:** CUDA is currently used during the training of deep network models to utilise the GPU and results in faster training of the models. This however leaves the CPU almost completely idle during the whole learning process. It would be feasible to have the CPU perform pre-processing tasks for current data with different parameter configurations or to render time-frequency transformations for new data while the networks are training. This would lead to optimal hardware usage and allow for greater experimentation in a shorter amount of time.

Automated Annotation

One problem encountered during the creation of this product was the fact that there isn't any available audio data that is annotated with instruments used at specific times. This has led to the network being trained on false

data, where there can be no guitar audio for a few seconds. This leads to biases being created for non-guitar audio and can affect validation results. With more time, a guitar recognition system could be created to isolate the guitar audio from the tracks, automatically annotating where guitar parts are found.
For this task I would propose using a VGG network architecture, a relatively new CNN architecture seeing large success in classifying image data. Although it is primarily used for images it has seen some success in classifying spectral representations audio data, particularly in noisy environments[8]. The VGG structure is very large comparatively and generally needs a lot of data. Freesound is an example of a repository of annotated audio data, taken from a variety of different environments, and would be ideal to train the network. The structure of the system would follow a similar system to the one created here: Extract data set from a repository, Apply necessary pre-processing (clipping, spectral transformations etc.), run through VGG network and validate.
After the network is trained to classify different instruments, it will then be used to detect whether there is a guitar at different audio slices and annotate appropriately, meaning those slices without be taken out of training and validation sets.
This would lead to unbiased model and be necessary for consequent future work into the subject.

Time-Frequency Representations

Although there has not been much difference between the time-frequency representations for this problem using the current data set. It is possible that when trying to identify non-folk artists or when trying to obtain precise transcription such as notes played and/or fret positions that the input data should be more optimised for musical analysis. It is proposed that the methods suggested below be implemented into the current pre-processing library, through functions similar to other time-frequency transformation methods:

- **CQCC –** Constant-Q Cepstral Coefficients are an effective way of representing data for musical analysis that address the shortcomings of MFCC. They are produced in a similar fashion to MFCC's, with the Constant-Q power spectrum used instead of Mel-Frequency. CQCC will generally take longer to calculate, as the Constant-Q transform produces a geometric frequency basis while DCT expects an orthogonal basis[4].
- **Dynamic-Q –** AnthemScore[5] is a piece of software for audio transcription into MIDI, that has achieved success in transcribing piano in particular. It uses a transformation they've penned "Dynamic Q". There is no available paper on Dynamic-Q; however it's been described as working by amplifying Q values in regions where nearby harmonics are detected.
  Increasing Q-Values has the trade-off increasing frequency resolution (So that detail about harmonics can be picked up more easily) at the cost of poorer time resolution.
  It is predicted that similar detail about audio harmonics maybe attained from using multiple constant-Q transformations with

different minimum frequencies and bin numbers as input into a CNN. This would work in a similar way to using RGB inputs for CNN's where each different constant-Q configuration would be a separate channel. Some experimentation with Constant-Q could create a significantly more robust model, and be potentially used in more complex problem spaces.

6.1.2 Applications into Other Work

Precise Guitar Transcription

Obtaining relevant semantic features from audio data has already been proven to be possible in this paper, along with others attempting to transcribe exact fret positions of guitar notes. Research has been completed in the area, but tunings are still required to be known before classification, and the research generally consists of one or few tunings used.
What is proposed here is that a solution is created that can obtain relevant semantic data such as: guitar type, exact fret positions and intonation of notes; from audio data in which we have no information on.
The methods laid out in Automatic Tablature Transcription of Electric Guitar Recordings Estimation of Score – and Instrument Related Parameters[6] are a good base for extracting exact fret position. The paper describes a solution for transcribing guitar tabs based on isolated guitar recordings; more importantly it also transcribes polyphonic guitar recordings. It works by having a lookup table of possible notes for a frequency and their positions based on the tuning. Once features are extracted from note onset, offset and pitch, they are passed through an SVM and plausibility filter to predict fret positions.
The plausibility filter effectively exploits known theory about guitar playing to reduce the amount of possible fret combinations and give weighting to the most probably position combinations. i.e. it may exclude combinations of notes where difference in fret position is greater than 4; and give more weight to positions that are closer together when notes are played quickly.
The main limitations of the solution proposed in this paper is that the guitar tuning must be known before transcription can be executed, and it requires knowledge on instrument construction for optimal results.
Combining the methods used here along with the system created in this project, it should be feasible to create a robust and thorough system that can transcribe unknown, non-annotated guitar recordings. This type of transcription opens the door to a plethora of further uses that have previously been very difficult tasks to complete, such as:

- **Capturing Style from Contemporary Music for Audio Synthesis**
- **Automatic Gamification of Guitar**
- **Cultural Preservation and Remastering of Damaged/Old Audio Tracks**

Capturing Style from Contemporary Music for Audio Synthesis

Solutions exist to synthesise music using AI from genres and artists commonly found in harmony textbooks, such as music from Bach or The Beatles. However, these already have a lot of theory behind them to be utilised to optimise the solutions, and in terms of classical music, things like the musical structure and tunings of instruments are very stagnant and are experimented with less often. To have a general system to capture style and utilise it in full automation for contemporary music containing real instruments is unseen.

By utilising a robust guitar transcription system, which would in theory be able to transcribe, to a certain accuracy, most given guitar recordings (enabled by the use of a tuning and instrument construction classifier), the ability to automatically annotate audio data with high detail is available.

Having a dataset so large, with precise annotations, allows a style dictionary to be formed from the data that has been processed. This opens the door for the use of dictionary learning approaches to be used to transfer style from one artist/genre to another. Although notes played may be the same, what decides the style of the piece is often the intonation of how the artist plays those notes and what effects are applied. There are already methods on how to apply an array of musical effects to audio, such as vibrato, tremolo, delay etc. to waveforms. The style transfer should work by taking a transcribed recording then find the best match between notes played between the recording and the style it needs to be transferred to. i.e. if you recorded C, D, E and wanted to transfer to Jimmy Hendrix' style, the system would search for the closest recorded data of in terms of timing and notes used by Jimmy Hendrix. Then some DSP techniques should be applied to add/subtract appropriate articulation that was used in Jimmy Hendrix's feature such as vibrato or sound properties such as changing an acoustic recording to sound like it was played on an electric. Important to note that Neural Style Transfer[9] has also yield good results in converted paintings and images into different art styles in recent years, and that a combination of both dictionary learning with some Neural Style Transfer applied for enhancement may provide optimal result. The worry with solely using Neural Style Transfer is that dissonance can be created easily within audio if small parts of the sound are adjusted incorrectly, so a more structured, predictable approach is desired.

Finally, using the semantic data and style banks created previously, there is the ability to synthesise completely new audio and restore possibly damaged audio. As seen in this project, and in many applications of CNN's throughout the audio processing field, computer vision techniques display much success when applied to spectrograms of audio. It is suggested that some adaptation of Inpaiting techniques (usually used for restoring old pictures or getting rid of items from images) are utilised to clean up old; damaged recordings. The ability to capture semantic data from new un-annotated recordings due to the transcription system, also means that the

inpainting techniques could be utilised as a creative tool. Artists could allow it to learn their style and clean-up their audio or fill in gaps/extra notes that they may not have thought of.

The techniques discussed here can also be applied more generally, to many different instruments and lay out a thorough design model for extracting and utilising semantic data in music.

Chapter 7

Reflection on Learning

Extracting guitar tunings from audio using machine learning techniques has been an invaluable learning experience across an array of domains. A large amount of time has been spent researching topics in the problem space of the project: DSP, Machine Learning, Deep Learning & CNNs. While having a novice level understanding of Machine Learning beforehand, the completion of this project given me a large amount of understanding and practice in the area and can see myself transferring those skills into non-DSP related fields with ease. By attempting to create a general use package as the foundations for my project, I can say that I've been able to apply the things learnt throughout my computer science career into a professional product and this has made me a lot more comfortable when handling Python as a whole.
As a result, the work done here has strongly developed my skills in researching concepts that are out of my comfort zone and allowed me to learn about niche topics specifically required for this field.

During this time, I have grown to appreciate the depth of feature extraction via signal processing and endeavour to go deeper with the skills I've learnt throughout. In all honesty, I feel extremely lucky to have delved into this research area, eagerly awaiting what future work I'll be involved in.

Bibliography

[1] Benjamin Blankertz. The Constant Q Transform.
http://doc.ml.tu-berlin.de/bbci/material/publications/Bla_constQ.pdf

[2] Furui, Sadaoki. (1981). Cepstral analysis technique for automatic speaker verification. Acoustics, Speech and Signal Processing
https://www.researchgate.net/publication/3176892_Cepstral_analysis_technique_for_automatic_speaker_verification

[3] A Complete Guide to K-Nearest -Neighbours with Applications in Python and R.
https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/

[4] Massimiliano Todsico, Hector Delgado, Nicholas Evans. (2017) Constant Q cepstral coefficients: A spoofing countermeasurefor automatic speaker verification
https://www.sciencedirect.com/science/article/pii/S0885230816303114

[5] AnthemScore
https://www.lunaverus.com/

[6] Christian Kheling Et. Al (2014) Automatic Tablature Transcription of Electric Guitar Recordings Estimation of Score – and Instrument Related Parameters
http://www.dafx14.fau.de/papers/dafx14_christian_kehling_automatic_tablature_trans.pdf

[7] Hagit Shatkay. (1995) The Fourier Transform - A Primer
https://pdfs.semanticscholar.org/409e/109c551767c296792ffa9f6d40a739c96ee7.pdf

[8] Shawn Hershey, Sourish Chaudhuri, Daniel P. W. Ellis, Jort F. Gemmeke, Aren Jansen, R. Channing Moore, Manoj Plakal, Devin Platt, Rif A. Saurous, Bryan Seybold, Malcolm Slaney, Ron J. Weiss, Kevin Wilson (2017)
https://arxiv.org/abs/1609.09430

[9] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge. (2015) A Neural Algorithm of Artistic Style
https://arxiv.org/abs/1508.06576

DIGITAL GUITAR TUNER
https://arxiv.org/ftp/arxiv/papers/0912/0912.0745.pdf