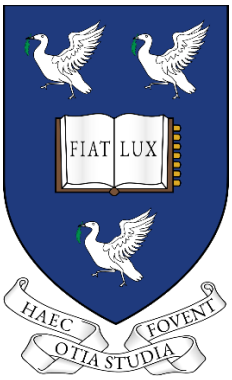
Machine Learning classification of Music Genres



Adam Thomas Kent

Supervisor: Dr. Rasmus Ibsen-Jensen

Department of Computer Science

University of Liverpool

This report is submitted for the degree of

'Master of Sciences'

MSc Computer Science September 2021

Abstract

Recognising music genres can be useful for sorting tracks by mood and similarity, allowing more control over large collections of digital music. This project features a multi-layer perceptron neural network designed to classify music audio data by its respective genre. The network shape features 3 hidden layers with Relu activation functions and a Softmax output layer. It is shown how music audio data can undergo feature extraction with the MFCC algorithm to obtain information on pitch and timbral differences, as would be recognised by the human ear, and how this data can be presented in a format suitable for the training of a neural network. Overfitting in the neural network is addressed by the employment of L2 regularisation and dropout, with the results showing a high level of accuracy, along with low categorical cross-entropy. This project could be useful for the future development of machine learning methods for music genre classification, perhaps with some commercial applications within existing music streaming services.

Word count: 7917 (excluding appendices).

Acknowledgements

I would like to thank my supervisor, Dr. Rasmus Ibsen-Jensen, for his support and kindness during this project.

Table of Contents

Introduction

The field of machine learning has given rise to many exciting applications of the subject. Such applications stem from various fields across the world of technology, from image recognition to AI chatbots, even to algorithmic trading within the stock markets.

This is a testament to the many different forms of machine learning now available, with supervised and unsupervised learning algorithms in use, and also the increasing prevalence of deep learning. As of the present year, this current age is often referred to as the Fourth Industrial Revolution (Sarker, 2021), where data is a key fuel, powering the modern engines of artificial intelligence. This data comes from a plethora of sources, given that now, many consumer electronics are connected to the internet, with our own personal mobile devices recording nearly all of our actions as data.

With all of this readily available data, tech companies are enabled to fine tune their software whereby features can be tailored to the individual user. This can produce personalised recommendations, but also seek and predict patterns in customer behaviour, particularly in e-commerce and entertainment platforms. Music streaming services such as Spotify and Apple music often present their users with "tailor-made" playlists or song recommendations which may have been formulated using artificial intelligence and data surrounding their activity within the streaming service.

Attempts are made to classify data in a wide variety of forms, starting with the conversion of data into a form readable by a machine learning system or artificial neural network (ANN). At the focus of this project has been the development of a machine learning model, designed to solve a multiclass classification problem of sorting music samples by their respective genres. This project was inspired by previous studies on speech and audio recognition, particularly those where training data has consisted of algorithmic calculations performed on signal samples. This project has also taken inspiration from studies on audio which has undergone analogue-to-digital sound conversion, with digital audio sampling forming the basis of the training data.

Some applications for this class of software could be implementation within existing music streaming services, for better management of music catalogues. It could also be used to aid the 'ripping' of music en masse from hard storage formats such as CDs, but also from analogue formats – such as Vinyl and Cassettes – through the use of digital recording devices, or analogue to digital conversion tools. This could be particularly useful with the latter example, as such audio files would not come with metadata regarding track information, such as the genre, or even the track name.

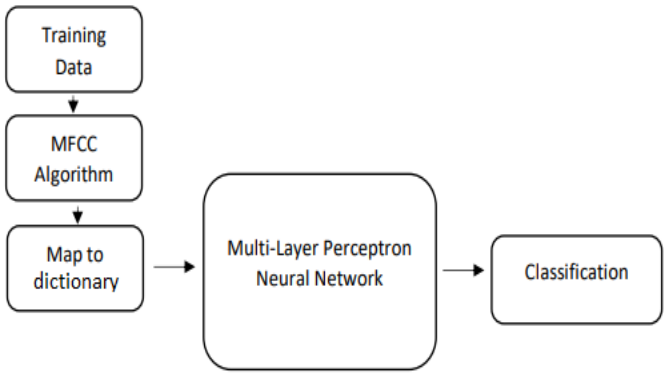


Figure 1: Schematic displaying potential functional steps for the software to be implemented

The architecture of this ANN was built using Python 3.9 (and external libraries), with steps laid out in the basic schematic on the previous page (Figure 1).

Firstly, a music dataset was acquired featuring samples of music from several genres. To make this data readable by the ANN, each sound file was be split into segments reflecting the sample rate, with the MFCC algorithm applied to each segment. This data was then written to a dictionary and stored in a CSV file.

A multi-layer perceptron was trained using the MFCC data stored in the dictionary, with 3 hidden layers, along with L2 regularisation and dropout to reduce overfitting of the MLP. It was found that the ANN could successfully predict the genre from which a 30 second sound file belonged to, with a high degree of accuracy after training. Whilst there is still room for improvement, these findings demonstrate just how complex forms of data such as audio can be manipulated and understood by some of the simplest of neural networks and machine learning techniques. This report will detail the research obtained during this project, alongside the steps of realising this concept, followed by a short evaluation and some concluding remarks.

Background

Audio recognition machine learning has advanced significantly in recent decades, particularly in the field of voice and speech recognition. To process input data for machine learning, various approaches have been used with variable degrees of success.

Digital audio is made up of sequences of samples with a sample rate which quantifies the number of samples per second, such as 44,100Hz in CD audio. A waveform (Figure 2) can help to represent changes in amplitude over time, for a specific length of audio. However, these representations do not provide information such as the frequency of each sample, and how spatially near samples can combine, to form elements of sound such as timbre.

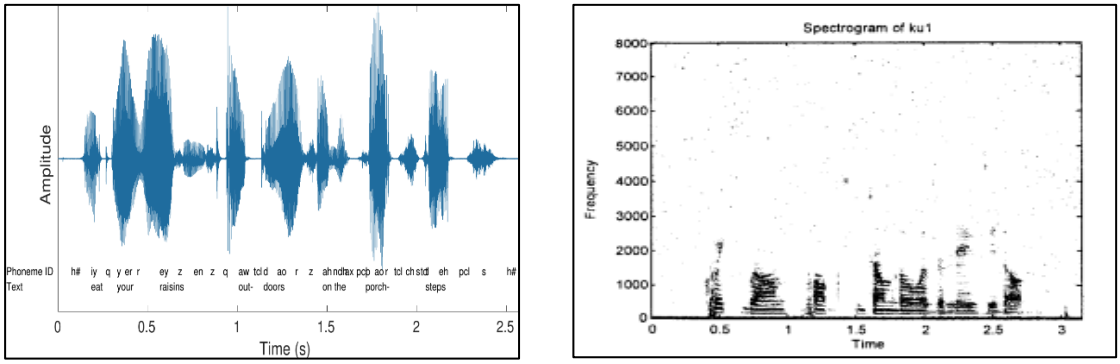


Figure 2: Left: A waveform (amplitude/time representation). From (Bäckström, 2020). Right: A greyscale spectrogram showing frequency information as well as amplitude/time information. From (Venayagamoorthy et al., 1998).

In previous studies on sound and speech recognition, Fourier transforms and Constant Q algorithms have been used for transforming raw audio data into frequency-time representations, such as a spectrogram. This process is known as feature extraction. A key algorithm underpinning these methods of sound data representation is the Fourier transform, as shown below:

$$X(k+1) = \sum_{n=0}^{N-1} x(n+1) W_N^{kn}$$

When this algorithm is applied to each sample in a raw audio file, it is called a Fast Fourier Transform and the results - when magnitude squared - can be plotted as a power spectrum. This representation shows the Power Spectral Density of a sound and has been used in previous studies on voice recognition (Venayagamoorthy et al., 1998). However, this is only a static interpretation of the original sound and does not disclose time information, thus, further steps are required in order to produce a spectrogram (see Figure 2), which features spatial time information.

To achieve this, the short time Fourier transform algorithm (STFT) has been used. These are calculated at different intervals within a given frame size, with the frame size (or window length) being a parameter which can vary. This will be expanded upon in the design section of this report. A spectrogram can be defined as the magnitude squared of the STFT values. Values are converted to logarithmic scale for decibels then the frequencies are mapped onto a greyscale image with varying degrees of brightness (Figure 2).

Additionally, in the field of musical genre classification, Cq-profiles have been used. These are 12 dimensional vectors calculated with the use of a constant Q filter bank (Purwins et al., 2000). These produce spectrograms similar to those made using short time Fourier transforms, however they are more useful in recognising pitch and the frequencies which correspond to these pitches.

In more recent studies, the Mel frequency cepstral coefficient (MFCC) algorithm has been used as feature representation in the neural network aided recognition of audio. This is a calculation which features a magnitude spectra projected to bands representing a set of frequencies, plotted logarithmically and compressed using a discrete cosine transformation (Purwins et al., 2019).

Using the MFCC algorithm produces data that better reflects elements within music, such as timbre as well as the standard pitch and amplitude elements, which appear in spectrogram representations.

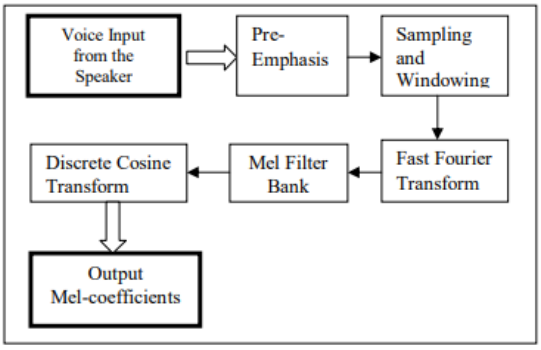


Figure 3: Example from speech recognition software of how to process raw audio data algorithmically. From (Chakraborty et al., 2014).

This algorithm has already been used in a significant number of studies for audio feature extraction, with the above figure outlining its usage in processing audio samples for speech recognition (see Figure 3). As this can be a time-consuming process, it would be beneficial to find Python libraries that could readily apply these algorithms and calculations on the training and input data used for this project.

A library which has been developed for audio manipulation and analysis is Librosa (McFee et al., 2021). It was decided that this library could be used as it has the MFCC algorithm built into it, to perform these calculations automatically. The documentation for this library is included in the bibliography at the end of this report.

A key requirement for this project is the sourcing of a suitable training dataset, there are several data sets that exist on the internet which have been used in previous studies. An example of this is the Latin music database (Silla et al., 2008), which contains 3160 songs in 10 genres, however these are limited to Latin American genres such as Tango and Bolero. Another dataset that is available online is the GTZAN Genre Collection, which was developed for a study in 2002 on musical genre classification (Tzanetakis and Cook, 2002). This contains more popular western genres such as Blues and Classical, therefore this represents a more suitable choice of dataset for this project. Other considerations such as copyright and data ethics will be reviewed in the next section of this report.

Ethical Use of Human Data

Data regulations can vary depending on the data in question, such as GDPR applying in Europe to personal data; Intellectual property law regarding inventions and patents; and Copyrights applied to Music, Film and other media. Whilst undertaking a project under an academic body, it is important to conform to all necessary data integrity requirements so that there are no breaches of any regulations.

As with most research projects concerning neural networks, a large set of data must be obtained to train the neural network. This data will be specific to the function that the neural network is intended to carry out, and thus the ethics surrounding the use of such data varies depending on the type of data. For instance, in a project where a neural network is being developed to segment medical images, a large set of medical imagery is warranted. This sort of data can be sensitive as it is usually acquired from patients, thus, sufficient consent or anonymisation is required in order to hide the patients' details.

No human data was used for the completion of this project, therefore there is little to be said regarding the ethical applications of such data in the production of this work.

In the case of this project, the data required consists of music samples. There are two approaches to obtaining a dataset of music samples whilst adhering to ethical use of data. The first approach would be to source music samples only from the public domain or the Creative Commons. Whilst there is a vast public domain of music, there can often be pitfalls associated with using the public domain. For instance, for a piece of music to be public domain, it has to be free of copyright. This means that most tracks falling under this category are substantially old and may predate the formation of certain genres.

There are several data sets that exist on the internet which have been used in previous studies. These include the aforementioned Latin music database (Silla et al., 2008), which contains 3160 songs in 10 genres, however these are limited to Latin American genres such as Tango and Bolero. The dataset that was used in this project is the GTZAN Genre Collection, which was developed for a study in 2002 on musical genre classification (Tzanetakis and Cook, 2002). This contains up to date and popular western genres such as Blues and Hip-hop, alongside some more conspicuous genre titles such as "Pop", however this still represented a suitable choice of dataset for this project.

This dataset has been used in a significant number of studies regarding music genre recognition, however there are some claims that the dataset lacks certain aspects of integrity. A study on the dataset performed by the University of Copenhagen (Sturm, 2012) found that for most instances of the dataset's usage, very few people had actually listened to its contents. They digress further to highlight inconsistencies in the data, such as reoccurring tracks and apparent lack of clarity regarding consent for the use of recordings. Nonetheless, this dataset has been used for many studies by reputable Universities without problems arising, with Sturm later highlighting that in the last 3 years (from 2013), more studies had used the dataset than in its first 8 years of availability (Sturm, 2014).

Still, with the discrepancies and faults pointed out in the above publications, in future studies it could be best to discard this option and to seek an alternative dataset.

Design

The architecture of this neural network can be summarised as of a series of steps, as previously laid out in the introduction and in Figure 1. Key tasks in the design of this software are: Selecting a dataset; Producing the training data; Building the neural network; and Testing the neural network.

The GTZAN music dataset, as mentioned in the background, was chosen as the source of training data for this project, however the chosen dataset must undergo a series of processing steps before it can be read as training data by a neural network. It is important that the initial input files are uniform. Therefore, if further sound files are to be added to the dataset, then they should be made uniform with the other files first. This can be accomplished by firstly, if necessary, padding and clipping the sound files to 30 seconds, and then resampling at 22050Hz if they are at a different sample rate, as this reflects how the files are presented in this particular dataset.

The Fourier transform algorithm, when applied to each sample over the duration of a sound file, is called a fast Fourier transform. This process takes amplitude data previously presented in the time domain and allows this information to be presented in the frequency domain. This can be described as breaking down the time-based sound signal data into a series of sine functions with their own magnitude, frequency and phase. If these individual sine waves are added together, then they should produce a representation similar in appearance to a waveform of the original file. When the amplitude of each sinusoidal term (calculated by a fast Fourier transform) is plotted against its frequency, the resulting representation is called a power spectrum. Some examples of this representation are shown below (Figure 4), which were calculated from two of the music files within the dataset.

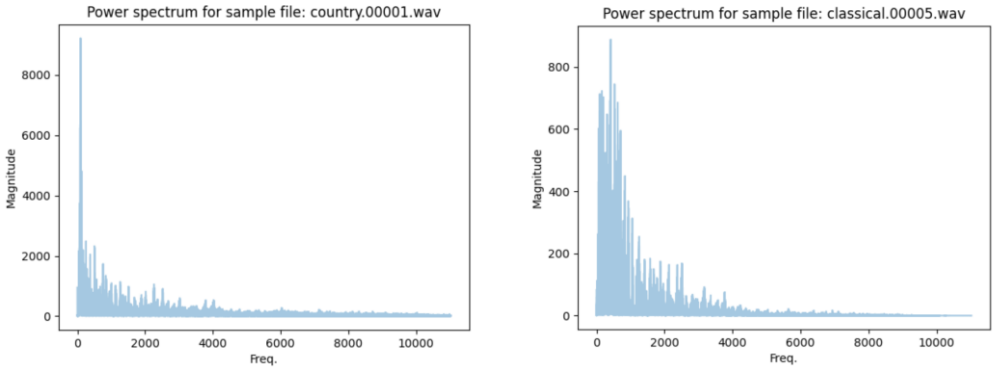


Figure 4: Power Spectrum representations of two example files from different genres, produced using the numpy, librosa and matplotlib libraries.

This is only a static interpretation of the associated sound file, as there is no time information provided in this sort of representation. Spectrograms, such as those in Figure 2, can provide time, frequency and amplitude information, all in one representation.

This sort of representation can be obtained using the short time Fourier transform algorithm, which features the Fourier transform being applied several times at different intervals, within a given frame size. This frame size, or window length is a parameter which can vary. For instance, a window length of 2048 samples is referred to as wideband window length, with narrowband being made up of 512 samples. To then produce a spectrogram, the amplitude data from these calculations is plotted on a logarithmic scale (for decibels) and then mapped onto a greyscale image, with varying degrees of brightness to represent the amplitude values of different frequencies over time.

Different forms of window can be utilised for the short time Fourier transform algorithm, with some accounting for the overlapping of segments. An example of this is the sliding Hann window, which has been used in previous studies to avoid spectral leakage (Huzaifah, 2017). The sliding Hann window is characterised by the formula displayed below:

$$X[n,k] = \sum_{m=0}^{L-1} x[m].w[n-m].e^{-i\frac{2\pi k}{N}m}$$

MFCC stands for MEL Frequency Cepstral Coefficients. These representations originate from speech recognition but have evolved into a prominent technique in in audio retrieval and processing. They represent timbral information (or the spectral envelope) of a signal by displaying the differences between neighbouring frequencies across a length of time. MFCCs have been successfully applied in the retrieval of timbre information by (Terasawa et al., 2012) in a study on speech recognition, and have also been applied in studies concerning the recognition of musical information.

Frequency data from the short time Fourier transform algorithm can be spread linearly, as it is in a standard spectrogram, or it can be mapped onto the Mel scale. The Mel scale is a scale where the units represent noticeably equal distances in perceived pitch. When humans perceive sound, studies have shown that we are better at detecting variation in lower frequencies than higher frequencies. For example, 400Hz and 900Hz may sound noticeably different, but the same

frequency difference in a higher set, such as 10400Hz and 10900Hz, may sound more similar to the human ear.

The frequency data can be mapped onto the Mel scale with 512 or 128 Mel bands. 512 is used when a wideband window length is utilised, and likewise 128 Mel bands for a narrowband window length. Studies have shown that wideband window lengths are better for longer varied sounds, and narrowband window length is better for shorter sounds or droning sounds with fewer frequency variations (Fernández Gallardo et al., 2014).

The librosa library used in this project builds upon NumPy, which already includes the fast Fourier transform algorithm as one of its many different functions. The librosa library also includes an MFCC function with numerous parameters such as the window length. Here we can see this in action whereby in this feature extraction plot, 1 set of MFCCs have been plotted (Figure 5).

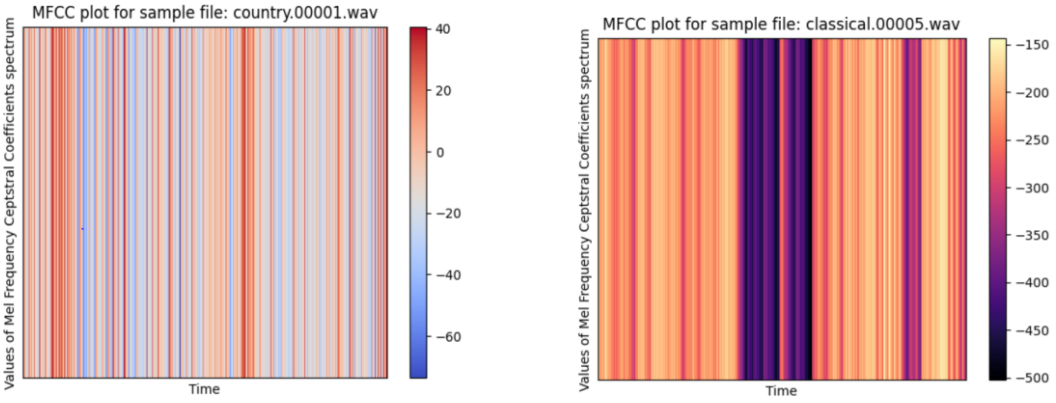


Figure 5: One set of MFCCs plotted for two example files using the librosa and matplotlib libraries. Note that the dark banding in classical.00005.wav indicates harsh fricative noise, perhaps a section of stringed instruments.

MFCC representations can be good for interpreting interactions between the source of a sound and any filters that have been applied to it (Sarria-Paja and Falk, 2017). For instance:

- Positive cepstral coefficient values indicate sonorant sounds, as most of the energy in these sounds is concentrated in the low-frequency regions.
- Negative cepstral coefficient values indicate fricative sounds, as most of the energy in these sounds is concentrated in the high frequency regions.

In standard practise, 10-20 MFCCs are used and have been used to represent the shape of a spectrum for a sound. (Mitrović et al., 2010). Here in this plot, you can see 12 MFCCS plotted to represent the whole sound more explicitly and clearly (Figure 6).

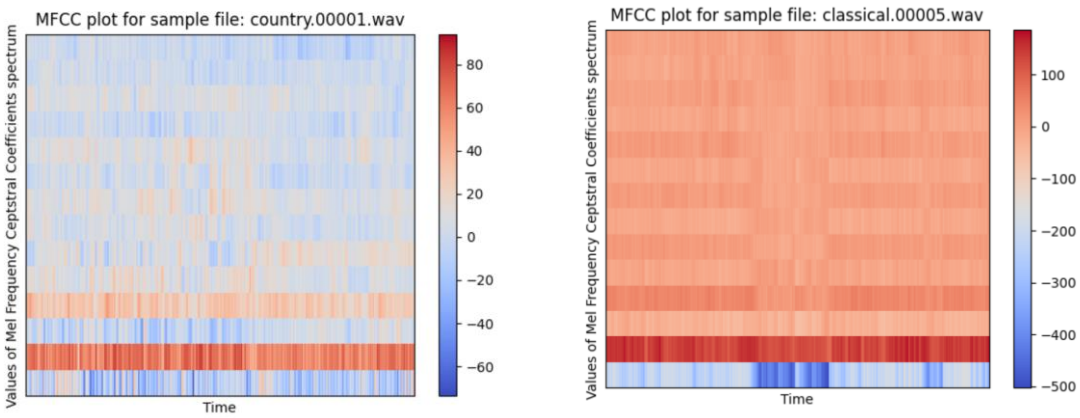


Figure 6: Fourteen sets of MFCCs plotted for two example files using the librosa and matplotlib libraries.

For the architecture of the neural network, it was decided that a multi-layer perceptron should be used. This type of neural network is an expansion of the original perceptron coined by Rosenblatt in 1961, as it allows for the solving of multi-class classification problems, such as the classification of musical recordings by their respective genres. The multi-layer perceptron (MLP) was built using the Keras interface, which runs on Google's open-source TensorFlow library.

The MLP is put together using the sequential model from Keras. There are 3 hidden layers with ReLU activation functions and respective weights of 256, 128 and 32. In the output layer, because we are dealing with a multi-class classification problem, a SoftMax activation function has been adopted, alongside a weight value of 10. SoftMax assigns probability values to each of the classes, or genres, in this multiclass classification problem. All of these values should reach a total of 1 when added together (Kabani and El-Sakka, 2016).

The 'Adam' optimiser built into the keras framework was also used in this neural network, which utilises stochastic gradient descent. With this, the learning rate parameter was fixed at 0.0001. After training the network, the accuracy and multi-class logarithmic loss (or categorical cross-entropy) were plotted in order to assess the performance of the neural network.

The training datasets for this neural network are presented in the form of NumPy arrays, with one array containing the MFCC data and one array containing sample information. To form the testing data, a library was used to randomly split the training data, producing a smaller subset of data for testing purposes, sized at 50% of the size of the original dataset. In the next section of this report, the initial findings and then subsequent steps to fine-tune the neural network will be detailed.

Realisation

The different stages of the project as shown in Figure 1 were divided into methods within the python code. The pre-processing method served to convert the audio files in the training dataset into arrays of numerical data representing the MFCC data/audio signals from the files. This would then serve as the training data for the model.

Outputs 1 – Processing of raw audio file data

```
for filename in filenames:
    filepath = os.path.join(path, filename)
    signal, sample_rate = librosa.load(filepath, sr=samplerate)

    for seg in range(totalsegments): #apply transformations to all segments of file
        first = avg_samps_per_segment * seg
        current = first + avg_samps_per_segment
        mfcc = librosa.feature.mfcc(signal[first:current], sample_rate, n_mfcc=num_mfcc, n_fft=numfastfourier, hop_length=hoplength)
        mfcc = mfcc.T #calculate the mfcc for the segment
        if len(mfcc) == exp_mfcc_perseg: #store mfcc only with expected number of mfcc per segment
            dataset["mfcc"].append(mfcc.tolist())
            dataset["code"].append(i-1)
```

Figure 7: Code snippet for enumerating through dataset and applying the MFCC algorithms to the raw audio files.

The sound data processing method enumerated through 50 files per genre, at 30 seconds long and with a sample rate of 22050Hz. For each of these tracks, data over the duration of the track was collected to produce an array of MFCC data. The librosa library was used for applying the MFCC algorithm using a set of previously defined parameters. The arrays produced by these calculations were then mapped to dictionaries and output into CSV files for use in the multi-layer perceptron class as training data.

Outputs 2 – Storing audio data in CSV format

This class would enumerate through each of the 5 files in each genre folder for the dataset. For each file, data would be stored in dictionary format under the following headings: Genre, Code, and MFCC. In this format, data under the MFCC heading would represent the MFCC values for the track.

Outputs 3 – Multi-layer Perceptron Neural Network

```
mlp = keras.Sequential([

    keras.layers.Flatten(input_shape=(X.shape[1], X.shape[2])), # input

    keras.layers.Dense(256, activation='relu'),        # hidden layer 1

    keras.layers.Dense(128, activation='relu'),        # hidden layer 2

    keras.layers.Dense(32, activation='relu'),        # hidden layer 3

    keras.layers.Dense(10, activation='softmax')        # output
])
optimizer = keras.optimizers.Adam(learning_rate=0.0001)
mlp.compile(optimizer=optimizer,
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

mlp.summary()
training = mlp.fit(X_input, y_input, validation_data=(X_test, y_test), batch_size=32, epochs=50)
```

Figure 8: Code snippet showing initial MLP architecture, prior to overfitting reduction measures being employed.

As laid out in the design section of this report, the multilayer perceptron consisted of 3 hidden layers, with ReLU activation functions, and an output layer utilising a SoftMax activation function.

When the entire dataset is used to train the neural network, and then a small subset of this data is used as test data, the network yields a high level of test accuracy (96.7%), with multi-class logarithmic loss also improving throughout the training process. However, this is a poor representation of how machine learning would be used in the real world, as in this instance, the testing data overlaps with the training data.

In order to more accurately simulate a real-world use of this software, the code was amended to utilise the sci-kit learn library again, this time splitting the whole dataset into separate, random, training and testing datasets, with less overlapping between the training data and testing data.

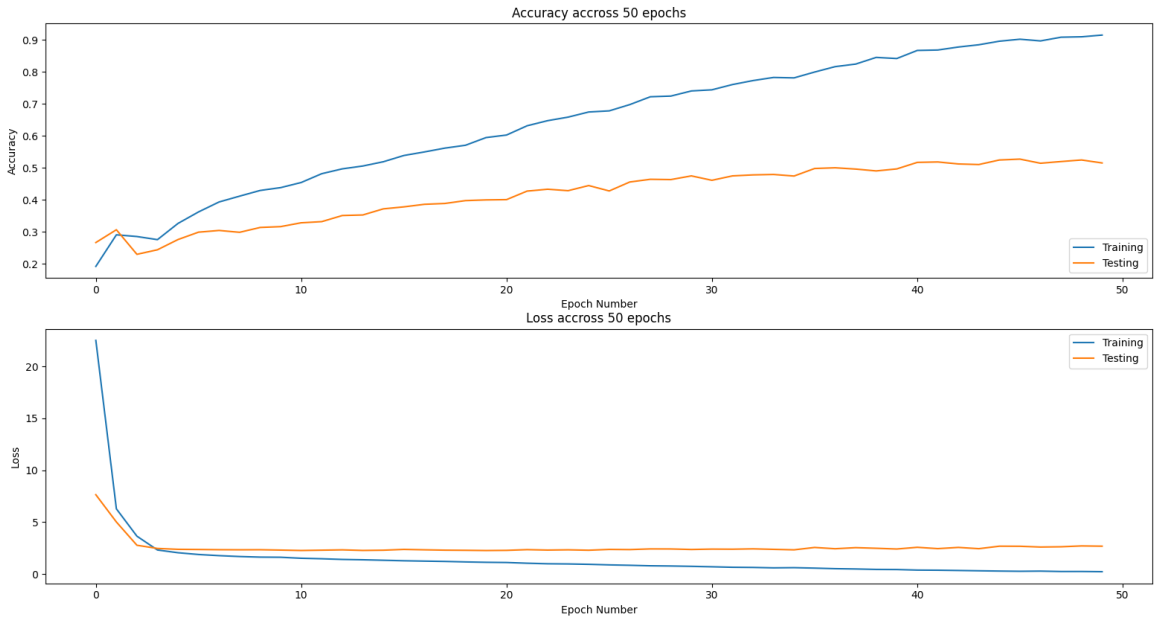


Figure 9: Initial Accuracy and Loss values over the training period, prior to overfitting reduction measures being employed.

With this structure in place, the neural network only produced an accuracy of 56.6 percent. On further inspection, the loss and accuracy for the testing process improved up until about 5 epochs through the training process, however a plateau then began to become. This pattern is one that is commonly seen in classification problems and is associated with over-fitting (Brownlee, 2019).

Outputs 4 – Reducing the overfitting problem

An interesting aspect of this project was discovering how certain methods in machine learning architecture can be employed to reduce the problems of overfitting. Through employment of dropout, and L2 regularisation applied via the Keras interface, the final accuracy produced from test data was around 70% after 300 epochs. The next few pages of this report will further detail how these measures were used to reduce the overfitting problem.

```python
def fixedModel(X_input, X_test, y_input, y_test):

    mlp = keras.Sequential([

        keras.layers.Flatten(input_shape=(X.shape[1], X.shape[2])), # input

        keras.layers.Dense(512, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),    # hidden layer 1
        keras.layers.Dropout(0.5),

        keras.layers.Dense(128, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),    # hidden layer 2
        keras.layers.Dropout(0.2),

        keras.layers.Dense(32, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)), # hidden layer 3
        keras.layers.Dropout(0.2),

        keras.layers.Dense(10, activation='softmax')    # output

    ])
    optimizer = keras.optimizers.Adam(learning_rate=0.0001)
    mlp.compile(optimizer=optimizer,
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])

    mlp.summary()
    training = mlp.fit(X_input, y_input, validation_data=(X_test, y_test), batch_size=32, epochs=300)
```

Figure 10: Code snippet showing MLP architecture, with dropout and regularisation employed to reduce overfitting.

The use of L2 regularisation and dropout has shown effectiveness in previous studies on multi-class classification, such as (Ulloa et al., 2018), where these techniques were used to improve a multi-layer perceptron designed to classify schizophrenia from MRI imaging.

Using regularisation reduces the prominence of large weights between neurons by adding penalties to the error functions (van Laarhoven, 2017). Dropout consists of randomly changing the number of neurons in the hidden layers between epochs, so that neurons are forced to adapt to work independently, which increases the robustness of the network (Phaisangittisagul, 2016). A probability of 50% dropout was applied to the first layer, in this network, with subsequent layers at 20% dropout probability, as can be seen in the code above. Because of the longer training time this causes, the number of epochs was also increased to 300. Lastly, the weight of the first layer was increased to 512, to compensate for the increased level of dropout added to it. Some other methods that could have been employed to reduce overfitting are as follows:

- Increasing the training dataset – this can be done via supplementation using artificial data or synthetic data.
- Reducing the number of hidden layers in the multi-layer perceptron, to mitigate over generalisation type effects of the network.
- Stopping training at fewer epochs – judging by time when optimal accuracy and loss levels are reached.

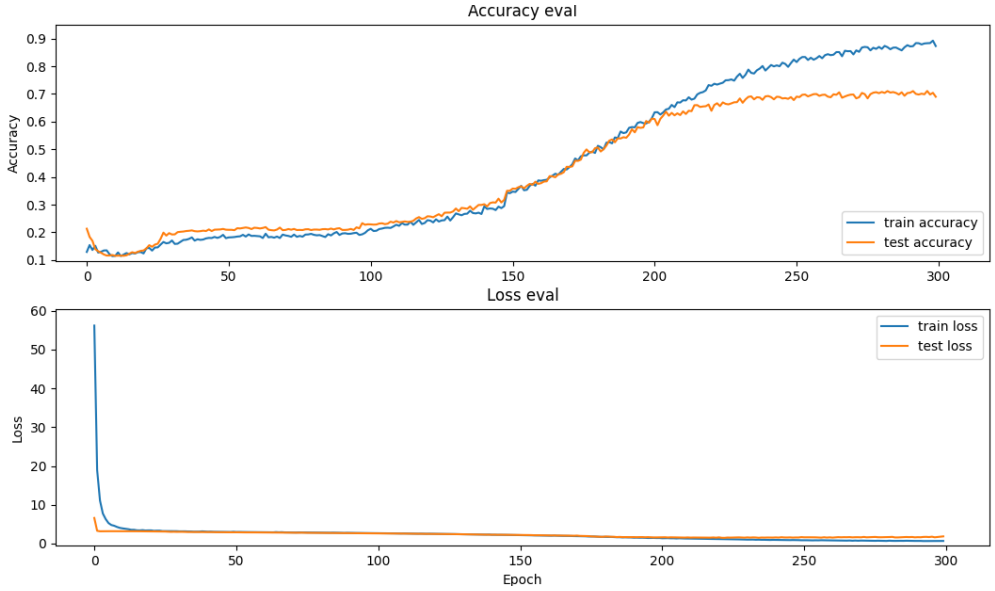


Figure 11: Final Accuracy and Loss values over the training period, with dropout and regularisation employed to reduce overfitting.

With the above methods applied, a maximum accuracy of around 70% was achieved in with the training data, compared with around 40% beforehand. To yield further accuracy, the dataset could have been made larger, through the use of artificial training data. Likewise, the training could have been stopped before the overfitting took place.

In Figure 12, we see the two graphs side by side for ease of comparison, here the effects of the added dropout and regularisation can be seen.

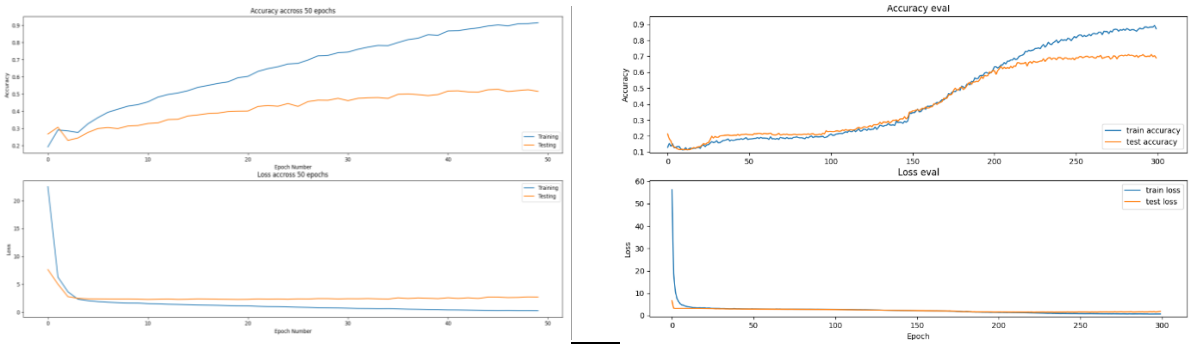


Figure 12: Comparison of Loss and Accuracy values before and after overfitting reduction measures were employed.

Evaluation

Initially, there had been a set out plan for the undertaking of this project. Firstly, the plan featured loose deadlines, which were finalised to actual dates. Unfortunately, there were a few setbacks in the project, however with the amended dates for each stage of the project, everything was still delivered successfully and on time. For instance, when overfitting problems arose during the testing stage, more time had to be allotted to researching into the reasons for overfitting and then finding out solutions for these problems.

There were also several deviations made to the project from the original plan. A notable difference between the proposed project and the final project is the use of the CSV format for the data dictionaries, rather than the JSON format as mentioned in the plan. The inspiration for this change came whilst working with CSVs for the export of laboratory data at my workplace. Here it became apparent that CSV represents a more streamlined format which can be displayed in excel for ease of reading if required. It can also be used with Python which features a CSV library, therefore this was adopted to replace the proposed JSON format for the training data inputs.

Whilst gathering training data, the Gutenberg public domain database was used as a source for some time ("Project Gutenberg," 2021). This database is rich in public domain material and represented a good way of collecting a large cohort of music tracks of a range of genres. Unfortunately, whilst attempting to collect sample data, constraints were realised related to the public domain nature of the database. For instance, for a piece of music to be public domain, it has to be free of copyright. This means that most tracks on the database are substantially old, with most predating the creation of certain genres. It was difficult, for instance, to source enough electronic music via this database to make up a substantial sample cohort.

As previously mentioned, when research has been performed on the GTZAN dataset, certain discrepancies such as the repetition of songs and potential genre overlap have been noted (Sturm, 2014). If the study was to be repeated, it could be a suitable idea to source an alternative dataset for means of producing the training data for the neural network.

Developing a suitable architecture for the multilayer perceptron took a lot of trial and error. To establish what parameters to use, such as the activation functions and number of layers, inspiration had to be taken from many previous studies on speech and audio processing. This is likewise for solving the overfitting problems with the neural network, in that many previous studies had to be consulted for reference in addressing such issues.

Despite the setbacks and longer times taken in the development of the project, a working model was achieved. Although the model could accurately classify the test data, only recycled data from the training set was used. Therefore, further studies are warranted to produce a model that can both suitably pre-process external audio data, and also classify it into its respective genre.

Finally, as with all machine learning methods, the use of more computational power, such as the employment of a device with a graphical processing unit could have improved the computational times of the model. Furthermore, with this improved hardware, other machine learning methods such as deep learning could potentially be employed to carry out a similar task in the future.

Learning Points

Throughout the duration of this project, there were improvements made in several areas of my skillset. The project was a test of resilience and along the way a lot of new things were learned.

1. 1 Skills and Knowledge: Fine-tuned python knowledge, building on other modules

Having undertaken the task in Python, this allowed me to gain substantial experience, building upon that previously acquired in other modules. Although there had been previous exposure to python, time constraints had often meant that there was little time to experiment with code and to truly grasp how certain features work. For example, the use of methods and passing arguments is something that was made use of during the presentation of this project, where different sound files were passed into graphical representation methods. This was accompanied by the use of for loops to aide the smooth running of the presentation where the next graph would appear in time for each data representation graph to be shown.

1.2 Acquired knowledge on physics of sound

Whilst learning about manipulations such as the Fourier transform and MFCCs, a breadth of knowledge on the science of sound waves and physics was acquired. This was fairly new to me, having not studied A level Physics or Mathematics, yet researching about how such calculations had been applied in the field of machine learning was something I found very interesting. Of particular interest was the implication that Power Spectrum graphs are comprised of smaller sinusoidal functions, or sine waves, which represent each digital sample in a sound file.

1.3 Appreciation of wide applications of neural networks

Whilst seeking a project idea, it became apparent just how many different formats of data can be processed for classification with neural networks. By performing mathematical transformations and calculations, data presented in many different forms, such as sound files and images, can be converted into numerical forms, which can then constitute the input data for neural networks and thus form machine learning solutions to classification problems.

1.4 Appreciation of Google's TensorFlow framework

Having used TensorFlow in previous group assignments, there was already a certain degree of familiarity with the framework. However, in this assignment, there was sufficient time to properly experiment with the framework and to gain a proper appreciation of it and its capabilities. This was achieved through having time to experiment with different settings, and to explore how supplementary frameworks such as Keras can be used to further enhance machine learning capabilities. Likewise, with Keras, I had had slight exposure to features such as the sequential model framework and built in optimizers. However, it was during this project where experimentation lead to a greater understanding of the framework than previously obtained.

1.5 Appreciation of Python libraries

With the code in this project utilising numerous libraries, it became apparent just how many functions can be implemented in Python. The built in libraries such as NumPy were able to greatly compliment external libraries such as librosa in the facilitation of easy mathematical calculations, saving time and improving the aesthetics of the code. Through realising the vast possibilities enable by the Python libraries, my inspiration has been taken outside of academia, where at work I have used libraries to complement my previous programming knowledge. An example of this is the use

of a Python library called beautifulsoup, which was utilised in an attempt to create a web scraping application. Furthermore, I have been made aware of the use of libraries such as IronPython for compatibility with the windows .NET framework. This could be used to link previous programming knowledge in VB.NET to the newer and current Python knowledge gained on this course.

2.1 Actions Critical to Success: Time taken to decide on project

During the start of this project, a substantial amount of time was taken to decide on a project area. Most of the recommended projects did not initially seem appealing to me, however, by not diving straight into one of these projects, the extra time was beneficial for deciding on a project concept which I found more interesting and was easier for me to engage with.

2.2 Learning about sound processing

With the concept of classifying sound files by genre, there was a great deal of understanding to be gained prior to designing the neural network architecture. A key task was to learn how such files could be processed into a form of data that could be used to train a neural network such as a multi-layer perceptron. Through reading previous studies on audio classification, I was able to learn about equations and algorithms such as MFCCs and Fourier transforms, which opened up a whole new area of machine learning to me which I had not previously read into. Paper such as "Voice recognition using MFCC algorithm" (Chakraborty et al., 2014b) and "Features for content based audio retrieval" (Mitrović et al., 2010) contained information on the use of these methods.

2.3 Improving overfitting in the model

With the initial multi-layer perceptron model yielding some substantial over-fitting, it was necessary to obtain information on how such overfitting could be mitigated. Through the finding of papers and the use of features built into the Keras framework/interface, I was able to apply overfitting reduction techniques to the neural network. Some of these techniques I had heard of before in previous group assignments, however it was in this assignment that I was able to learn more about these processes through changing their parameters in a trial-and-error fashion. By doing this, I was able to tweak the neural network in order to obtain results which yielded less overfitting than the initial model.

3.1 Things I would do differently: Making a more informed choice on training data

As highlighted in this report, the dataset that was chosen had in fact been subject to substantial levels of scrutiny. If the project was to be repeated in the future, more due diligence would be warranted regarding the choice of training data, in order to avoid the use of datasets which have been subject to various levels of criticism.

3.2 Better time management and planning of the project

A large pitfall in the execution of this project was poorly managed time and the constraints placed on project progress by external factors. These factors, including the timing of resit exams, were poorly accounted for by myself. Time could have been managed better to facilitate more simultaneous progress in both of these areas, however, with the aide of extensions provided by the department, the project was still able to go ahead and meet newly reassigned deadlines.

3.3 Testing with external data.

If I had developed this project further, I could have provided external data for the testing of the model (ie. Music from outside of the used dataset). These tracks would first have had to be processed to match the quality and sample rate of the files within the dataset, however, doing this could have enabled an accuracy comparison between classifying internal testing data versus external testing data. This could have allowed me to better demonstrate the capabilities of this neural network.

3.4 Different or more modern machine learning models.

The multi-level perceptron model that was demonstrated in this project represents a fairly basic machine learning model, drawing from the original perceptron model which was published in 1961. Although this project utilised present computational power and modern programming languages, the concept for the neural network architecture is somewhat well known. In future studies, more modern machine learning models could potentially be used such as those making use of deep learning or convolutional layers. Additionally, more elements from the certain modules taught by the school could have been utilised, such as Genetic Algorithms, which form part of one of the modules that can be taken as part of the MSc Computer Science masters programme.

Professional Issues

Having completed this project, it is useful to visualise how the undertaken work fits corresponding to the British Computer Society (The Chartered Institute for IT) code of conduct.

The four principles outlined in the code consist of:

1. *You make IT for everyone.*

This refers to professionalism, regard for privacy and behaviour towards others.

2. *Show what you know, learn what you don't.*

This refers to competence and a dedication to continued learning within the field of IT.

3. *Respect the organisation or individual you work for.*

This refers to taking the correct level of responsibility for your work and carrying such responsibilities with a high level of care.

4. *Keep IT real. Keep IT professional. Pass IT on.*

This last point refers to the maintenance of good standards and a good public image as a representative of the society.

The software developed during this project, whilst still in a primitive stage, could potentially have commercial applications, therefore the principles outlined in the code should be adhered to and considered during the stages of the software development. A key sub-principle within the first principle relates to the legitimate right of third parties. This could be in regard to intellectual property claims or copyright claims, with the BCS code expecting full cooperation of the software

developer. In this project, research was undertaken into sources of training data which would not infringe the rights of third parties. If the software was to be developed further, it would be important to make sure that elements of the architecture do not infringe on the creations of other individuals or companies who may have already developed similar architecture.

The second principle refers to continued development of skills during the undertaking of a project, and to address any gaps in knowledge accordingly before carrying out a task. This project has featured learning at multiple steps during the software development. Many times it has been necessary to go away and learn about a certain concept before attempting to implement it. An example of this are the methods used to mitigate overfitting of the neural network model, where new knowledge had to be acquired regarding regularization and network dropout, before attempting to tweak the parameters of these methods. As this work was not carried out on behalf of a company, there was less concern regarding potential damage of company reputation or liabilities of this manner, however, if this software was to be adapted for commercial use, it would be useful to take heed of such sub-principles.

Again, as with the previous two principles, as the software is not currently being commercially developed, the principle of duty to relevant authority is less significant. However, as this is a project being carried out under the supervision of an academic supervisor, there is a certain level of duty to communicate clearly and to fairly acknowledge any support received from them during the project. As a student of the University, it is also good to remember that they like to see a high standard of work produced, in order to to help represent them well as an independent academic institution.

The last principle refers to being a good ambassador for the IT industry and the British Computing Society (The Chartered Institute for IT). If I was a member of this professional body, I would ensure that the project complied with all of these principles, to uphold the good reputation of the BCS. The work undertaken during this project has represented other computer science research in a good light, drawing inspiration from previous studies and learning new information from them. It is important to respect the work of other people in the field, so that we can learn from each other. This is of particular importance in the area of open-source software, where innovation is carried out by cooperation of multiple people in the field of computer science and beyond. The bulk of the Python libraries used for this project were developed using open-source methods and models. Popular machine learning frameworks such as TensorFlow are also open source, which heavily rely on the contributions of and cooperation from people who are situated all over the world.

Conclusions

Multi-class classification problems represent a large amount of the applications of machine learning, particularly in the fields of speech recognition and autonomous systems. Whether the classifiable data represents image features or sound features, there have been many solutions developed in recent times as new innovations in artificial intelligence arise.

Despite the new forms of machine learning that are currently available, this project demonstrated that a model as simple as the multi-layer perceptron could still be effective in solving this problem. This project also explored, drawing on previous studies, how sound can be processed into a form that can be used to train neural networks and address such classification problems.

Wideband window length MFCC features were produced which enabled the network to characterise genres, despite the many similarities that can exist between genres such as rock and metal, or jazz and classical.

Kernel L2 regularisation with a lambda value of 0.001, as has been employed in previous multi-class classification problems, contributed to less overfitting in the network alongside the addition of 20% dropout probability, which helped to build a more robust neural network.

Further studies are warranted to explore how more recent innovations in machine learning could be applied to this same problem, such as deep learning. As discussed, a more reliable dataset could also be used in future studies to produce a more effective and widely applicable machine learning system.

Bibliography

Bäckström, T., 2020. Waveform. Introd. Speech Process., Aalto University Wiki. URL https://wiki.aalto.fi/display/ITSP/Waveform

Brownlee, J., 2019. Multi-Label Classification of Satellite Photos of the Amazon Rainforest. Mach. Learn. Mastery. URL https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-satellite-photos-of-the-amazon-rainforest/ (accessed 9.30.21).

Chakraborty, K., Talele, A., Upadhya, S., 2014a. Voice recognition using MFCC algorithm. Int. J. Innov. Res. Adv. Eng. IJIRAE 1, 2349–2163.

Chakraborty, K., Talele, A., Upadhya, S., 2014b. Voice Recognition Using MFCC Algorithm. Nter Natl. Jour Nal Nnovative Resear Ch Adv. Eng. Ing 1, 5.

Fernández Gallardo, L., Wagner, M., Möller, S., 2014. Advantages of Wideband over Narrowband Channels for Speaker Verification Employing MFCCs and LFCCs, Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH. https://doi.org/10.21437/Interspeech.2014-286

Huzaifah, M., 2017. Comparison of Time-Frequency Representations for Environmental Sound Classification using Convolutional Neural Networks. ArXiv170607156 Cs.

Kabani, A., El-Sakka, M., 2016. Object Detection and Localization Using Deep Convolutional Networks with Softmax Activation and Multi-class Log Loss. https://doi.org/10.1007/978-3-319-41501-7_41

McFee, B., Metsai, A., McVicar, M., Balke, S., Thomé, C., Raffel, C., Zalkow, F., Malek, A., Dana, Kyungyun Lee, Nieto, O., Ellis, D., Mason, J., Battenberg, E., Seyfarth, S., Yamamoto, R., Viktorandreevichmorozov, Keunwoo Choi, Moore, J., Bittner, R., Hidaka, S., Ziyao Wei, Nullmightybofo, Hereñú, D., Fabian-Robert Stöter, Friesch, P.,

Weiss, A., Vollrath, M., Taewoon Kim, Thassilo, 2021. librosa/librosa: 0.8.1rc2. Zenodo. https://doi.org/10.5281/ZENODO.4792298
Mitrović, D., Zeppelzauer, M., Breiteneder, C., 2010. Features for Content-Based Audio Retrieval, in: Advances in Computers. Elsevier, pp. 71–150. https://doi.org/10.1016/S0065-2458(10)78003-7
Phaisangittisagul, E., 2016. An Analysis of the Regularization Between L2 and Dropout in Single Hidden Layer Neural Network. 2016 7th Int. Conf. Intell. Syst. Model. Simul. ISMS. https://doi.org/10.1109/ISMS.2016.14
Purwins, H., Blankertz, B., Obermayer, K., 2000. A New Method for Tracking Modulations in Tonal Music in Audio Data Format. pp. 270–275 vol.6. https://doi.org/10.1109/IJCNN.2000.859408
Purwins, H., Li, B., Virtanen, T., Schlüter, J., Chang, S., Sainath, T., 2019. Deep Learning for Audio Signal Processing. IEEE J. Sel. Top. Signal Process. 13, 206–219. https://doi.org/10.1109/JSTSP.2019.2908700
Sarker, I.H., 2021. Machine Learning: Algorithms, Real-World Applications and Research Directions. SN Comput. Sci. 2, 160. https://doi.org/10.1007/s42979-021-00592-x
Sarria-Paja, M., Falk, T.H., 2017. Variants of mel-frequency cepstral coefficients for improved whispered speech speaker verification in mismatched conditions, in: 2017 25th European Signal Processing Conference (EUSIPCO). Presented at the 2017 25th European Signal Processing Conference (EUSIPCO), IEEE, Kos, Greece, pp. 91–95. https://doi.org/10.23919/EUSIPCO.2017.8081175
Silla, C., Koerich, A., Kaestner, C., 2008. The Latin Music Database. pp. 451–456.
Sturm, B.L., 2014. The GTZAN dataset: Its contents, its faults, their effects on evaluation, and its future use. J. New Music Res. 43, 147–172. https://doi.org/10.1080/09298215.2014.894533
Sturm, B.L., 2012. An analysis of the GTZAN music genre dataset, in: Proceedings of the Second International ACM Workshop on Music Information Retrieval with User-Centered and Multimodal Strategies, MIRUM '12. Association for Computing Machinery, New York, NY, USA, pp. 7–12. https://doi.org/10.1145/2390848.2390851
Terasawa, H., Berger, J., Makino, S., 2012. In Search of a Perceptual Metric for Timbre: Dissimilarity Judgments among Synthetic Sounds with MFCC-Derived Spectral Envelopes. J. Audio Eng. Soc. 60, 674–685.
Tzanetakis, G., Cook, P., 2002. Musical genre classification of audio signals. IEEE Trans. Speech Audio Process. 10, 293–302. https://doi.org/10.1109/TSA.2002.800560
Ulloa, A., Plis, S., Calhoun, V., 2018. Improving Classification Rate of Schizophrenia Using a Multimodal Multi-Layer Perceptron Model with Structural and Functional MR. ArXiv180404591 Cs.
van Laarhoven, T., 2017. L2 Regularization versus Batch and Weight Normalization. ArXiv170605350 Cs Stat.
Venayagamoorthy, G.K., Moonasar, V., Sandrasegaran, K., 1998. Voice recognition using neural networks, in: Proceedings of the 1998 South African Symposium on Communications and Signal Processing-COMSIG '98 (Cat. No. 98EX214). Presented at the Proceedings of the 1998 South African Symposium on Communications and Signal Processing-COMSIG '98 (Cat. No. 98EX214), pp. 29–32. https://doi.org/10.1109/COMSIG.1998.736916

Appendix i - Documentation for libraries used:

https://librosa.org/doc/latest/index.html

https://keras.io/api/

https://www.tensorflow.org/api_docs

https://matplotlib.org/

https://scikit-learn.org/stable/

Appendix ii – Source code

1) extractingdata.py

```python
import csv
import os
import math
import librosa, librosa.display
import matplotlib.pyplot as plt
import numpy as np

def powerspectrum(file):
    ''' Loads the file for manipulation and perform fast fourier transforms, Calculate magnitude and frequency for power spectrum plot.
        :file : The sample file to undergo processing.
    '''

    signal, sample_rate = librosa.load(file, sr=22050)
    Fft = np.fft.fft(signal) #fourier transform
    spec = np.abs(Fft)  #magnitude calculation
    f = np.linspace(0, sample_rate, len(spec)) #frequency calculation

    halfspec = spec[:int(len(spec)/2)] #divide magnitude /2
    halffreq = f[:int(len(spec)/2)] #divide freq /2

    plt.figure()
    plt.plot(halffreq, halfspec, alpha=0.4)
    plt.ylabel("Magnitude")
    plt.xlabel("Freq.")
    plt.title("Power spectrum for sample file: " + file)

    plt.show()

def mfccplot(file, n_mfcc):
    ''' Perform short time fourier transforms then plot MFCC representations.
    :file : The file to be represented
    :n_mfcc: The number of MFCCs to be represented in the plot produced by this method.
    '''
    hoplength = 1024 #half of wideband window length
    numfastfourier = 2048 # window in num. of samples. Wideband window length
    signal, sample_rate = librosa.load(file, sr=22050)
    MFCCs = librosa.feature.mfcc(signal, sample_rate, n_fft=numfastfourier, hop_length=hoplength, n_mfcc=n_mfcc)

    plt.figure()
    librosa.display.specshow(MFCCs, sr=sample_rate, hop_length=hoplength)
    plt.ylabel("Values of Mel Frequency Ceptstral Coefficients spectrum")
    plt.xlabel("Time")
    plt.colorbar()
    plt.title("MFCC plot for sample file: " + file)
    plt.show()
```

```python

def processmfccs(path_data, csv_path, num_mfcc=13, numfastfourier=2048, hoplength=1024, totalsegments=5):
    """Performs calculations on dataset of MFCC values and saves with genre code into CSV file
    """
    fieldnames  = ["map", "code", "mfcc"]
    dataset = {"map": [], "code": [], "mfcc": []} #dataset stored as dictionary

    avg_samps_per_segment = int(totalsamps / totalsegments)
    exp_mfcc_perseg = math.ceil(avg_samps_per_segment / hoplength)


    for i, (path, folder, filenames) in enumerate(os.walk(path_data)):
        if path is not path_data:
            nameofgenre = os.path.split(path)[-1] #sorts by genre name
            dataset["map"].append(nameofgenre)
            print("\nCurrently processing genre: " + nameofgenre)

            for filename in filenames:
                filepath = os.path.join(path, filename)
                signal, sample_rate = librosa.load(filepath, sr=samplerate)

                for seg in range(totalsegments): #apply transformations to all segments of file
                    first = avg_samps_per_segment * seg
                    current = first + avg_samps_per_segment
                    mfcc = librosa.feature.mfcc(signal[first:current], sample_rate, n_mfcc=num_mfcc, n_fft=numfastfourier, hop_length=hoplength)
                    mfcc = mfcc.T #calculate the mfcc for the segment
                    if len(mfcc) == exp_mfcc_perseg: #store mfcc only with expected number of mfcc per segment
                        dataset["mfcc"].append(mfcc.tolist())
                        dataset["code"].append(i-1)

```

```
    writer = csv.DictWriter(open(csv_path, "w"), fieldnames=fieldnames) # save to csv file
    writer.writerow(dict(dataset))

path_data = 'C:/Users/Adam/Desktop/Project/Data/'
csv_path = "data_10.csv"
samplerate = 22050
songlength = 30 # measured in seconds
totalsamps = samplerate * songlength

if __name__ == "__main__":
    processmfccs(path_data, csv_path, totalsegments=10)
    files = ["country.00001.wav", "hiphop.00046.wav", "classical.00005.wav", "rock.00025.wav"]
    for file in files:
        powerspectrum(file)
    for file in files:
        mfccplot(file, 1)
    for file in files:
        mfccplot(file, 14)
```

2) multilayerp.py

```
import json
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import matplotlib.pyplot as plt

path_data = "C:/Users/Adam/Desktop/Project/data_10.csv"

def getdeta(path_data):

    with open(path_data, "r") as fp:
        data = json.load(fp)

    X = np.array(data["mfcc"]) # list to array
    y = np.array(data["code"]) # list to array
    print("Loaded dataset")
    return  X, y

def overfittingModel(X_input, X_test, y_input, y_test):

    mlp = keras.Sequential([

        keras.layers.Flatten(input_shape=(X.shape[1], X.shape[2])), # input

        keras.layers.Dense(256, activation='relu'),        # hidden layer 1

        keras.layers.Dense(128, activation='relu'),        # hidden layer 2

        keras.layers.Dense(32, activation='relu'),        # hidden layer 3

        keras.layers.Dense(10, activation='softmax')        # output
    ])
    optimizer = keras.optimizers.Adam(learning_rate=0.0001)
    mlp.compile(optimizer=optimizer,
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])

    mlp.summary()
    training = mlp.fit(X_input, y_input, validation_data=(X_test, y_test), batch_size=32, epochs=50)

    ig, axs = plt.subplots(2)

    axs[0].plot(training.history["accuracy"], label="Training") # plot accuracy
    axs[0].plot(training.history["val_accuracy"], label="Testing")
    axs[0].set_ylabel("Accuracy")
    axs[0].set_xlabel("Epoch Number")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy accross 50 epochs")
```

```
49
50      axs[1].plot(training.history["loss"], label="Training") # plot loss
51      axs[1].plot(training.history["val_loss"], label="Testing")
52      axs[1].set_ylabel("Loss")
53      axs[1].set_xlabel("Epoch Number")
54      axs[1].legend(loc="upper right")
55      axs[1].set_title("Loss accross 50 epochs")
56      plt.show()
57
```

```
58  def fixedModel(X_input, X_test, y_input, y_test):
59
60      mlp = keras.Sequential([
61
62          keras.layers.Flatten(input_shape=(X.shape[1], X.shape[2])), # input
63
64          keras.layers.Dense(256, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),    # hidden layer 1
65          keras.layers.Dropout(0.2),
66
67          keras.layers.Dense(128, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)),    # hidden layer 2
68          keras.layers.Dropout(0.2),
69
70          keras.layers.Dense(32, activation='relu', kernel_regularizer=keras.regularizers.l2(0.001)), # hidden layer 3
71          keras.layers.Dropout(0.2),
72
73          keras.layers.Dense(10, activation='softmax')    # output
74
75      ])
76      optimizer = keras.optimizers.Adam(learning_rate=0.0001)
77      mlp.compile(optimizer=optimizer,
78                  loss='sparse_categorical_crossentropy',
79                  metrics=['accuracy'])
80
81      mlp.summary()
82      training = mlp.fit(X_input, y_input, validation_data=(X_test, y_test), batch_size=32, epochs=150)
83
84      fig, axs = plt.subplots(2)
85
86      axs[0].plot(training.history["accuracy"], label="train accuracy") # plot accuracy
87      axs[0].plot(training.history["val_accuracy"], label="test accuracy")
88      axs[0].set_ylabel("Accuracy")
89      axs[0].legend(loc="lower right")
90      axs[0].set_title("Accuracy eval")
91
92      axs[1].plot(training.history["loss"], label="train loss") # plot loss
93      axs[1].plot(training.history["val_loss"], label="test loss")
94      axs[1].set_ylabel("Loss")
95      axs[1].set_xlabel("Epoch")
96      axs[1].legend(loc="upper right")
97      axs[1].set_title("Loss eval")
98
99      plt.show()
100
101     predictX = X_test[50]
102     predicty = y_test[50]
103     predictX = predictX[np.newaxis, ...] # extra dimension for input data
104     guess = mlp.predict(predictX) # attempt prediction
105     # get index with max value
106     maxvalueindex = np.argmax(guess, axis=1) # select highest value
107     print("Target: " + str(predicty) + "Predicted genre code: " + str(maxvalueindex))
108
109
110 if __name__ == "__main__":
111
112     X, y = getdeta(path_data)
113     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)
114
115     #overfittingModel(X_train, X_test, y_train, y_test)
116     fixedModel(X_train, X_test, y_train, y_test)
```