Convolutional Neural Network to Classify Electric Guitar Distortions

Maxime Greffe - 40430624

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BEng (Hons) Software Engineering

School of Computing

April 21, 2021

Authorship Declaration

I, Maxime Greffe, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed my- self;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

Signed: Maxime Greffe

Date: April 2021

Matriculation Nº: 40430624

Data Protection Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.
Maxime Greffe

The University may not make this dissertation available to others.

Abstract

Convolutional Neural Networks have been a popular way to classify images in recent years. As a form of computer vision, it has been shown that they can be used to recognize everyday objects or handwritten text. This paper aims to find out if Convolutional Neural Networks can be used to classify audio files, and more particularly electric guitar distortions. Throughout a well-focused literature review and experiments, this project analyses the effectiveness of such networks on visual representations of audio samples.

It has been found that, when using the correct visual representation, Convolutional Neural Networks can indeed identify and classify different types of guitar distortions. While such outcome requires an important quantity of data, this project opens the door to new ways to classify audio files.

In order to demonstrate the efficiency of the trained model, a prototype application has been developed. It was found that further work on this prototype could lead to a very efficient way to train new models on multiple guitar amplifiers and allow guitar players to analyse their own sounds in order to find out which amplifier settings are required to achieve a similar distortion.

Contents

List of Figures

List of Tables

1 Introduction

It has never been easier to learn and play music than it is now. While music was for a long time reserved to privileged individuals that had followed a musical education, Chadabe (1997) declared that "Interactivity at home means that an amateur, perhaps without talent or skill, can participate in a rewarding way in a musical process". Amateur music having a huge market potential, it has evolved throughout the years and the sheer amount of resources available on the internet, along with free and open-source software and the mass production of affordable musical instruments and devices has turned it into a common activity. Amongst musicians, guitar players are some of the most common. The great choice of equipment available, from the different guitar brands and makes, the standalone effects pedals or boards and the huge variety of amplifiers, can make it a difficult/expensive task to find the right sound. Although many guitarists like to shape and develop their own sound, they are often inspired by guitar parts on a record they like.

1.1 Problem Statement

Artificial intelligence (AI) allows computers to reproduce or at least to mimic the reasoning and decision-making of the human mind. According to Russel & Norvig (2018), "AI is relevant to any intellectual task; it is truly a universal field". It is not surprising that a wide variety of problems are attempted to be solved by the means of AI. It can be applied to any project that requires some degree of thinking and can, under the right conditions, outperform human beings in solving problems efficiently, often by performing complex operations and calculations faster that a human being is capable of, but also by removing the chance for human mistakes.

AI seems particularly adapted to the problem at hand, which essentially revolves around recognising, analysing and classifying audio files.

To be able, via the means of a neural network, to identify and classify such sounds would greatly improve the sound making process, or at least provide

some useful insight in giving the user the recipe (i.e. settings) needed to achieve a similar sound. Such software would fit very well in amateur environments where musicians are more likely to have a limited access to different types of equipment.

1.2 Research Questions

This project aims to answer the following questions:

- Can deep learning and neural networks recognize and classify different guitar sounds?
- Can visual representations of audio spectrums be used by a convolutional neural network to identify different types of guitar distortions?
- Can a graphical user interface be produced to allow users to classify their own audio files?

1.3 Aims and Objectives

The aim of this project is to develop a software that will allow for guitar distortion recognition and classification, providing the user with the needed settings to achieve a desired sound.

The objectives of this project are listed below:

1. Review the existing literature on sound recognition and classification
2. Review the existing literature on deep learning and neural networks
3. Create a dataset that can be used to train a CNN
4. Implement a convolutional neural network that will use visual representations of audio signals to classify them
5. Implement a graphical user interface to make the software easy to use
6. Evaluate the quality of the produced output

1.4 Scope and Limitations

This project will focus on identifying, classifying and reproducing the sound of one specific amplifier, the Kuassa Amplikation Lite. This amplifier has been chosen as it is a virtual amplifier (i.e. a software than runs on computer) available for free, which will make the testing part easier as anyone with a computer should be able to use the program. Should the project be successful and progress faster than expected, the exercise could be extended to use other amplifiers.

2 Literature Review

2.1 Classifying guitar sounds based on distortion

This section provides some background on guitar distortion and sound recognition and classification, as well as the types of visual audio representations that may be use to extract features.

2.1.1 Guitar Distortions

The design of the first guitar amplifiers would create a distortion when the volume was increased beyond the load that they could handle (Rubin, 2007). Guitarists found this sound quite harmonic and decided to experiment around it, leading to the apparition of blues rock and the design of amplifiers that would produce a nice distortion. Nowadays, a large number of devices are available to produce a distortion, ranging from cheap to expansive, analogue to digital, as external guitar pedals plugged between the guitar and the amplifier or as part of the amplifier. Each device comes with its own settings options, the most common ones being gain/level, tone, and some extra controls depending on the device.

There are two general types of distortions. For the sake of clarity, they will be called *overdrive distortion* and *plain distortion*, but it is worth noting that guitarists refer to them only as overdrive and distortion. Overdrive distortions either provide a huge volume boost in order to overdrive the valves in a tube amplifier, which is essentially how the sound was obtained in the early days of distortions (and is still in use today), or mimic the effect of over-driven valves within a tube amplifier. A plain distortion tends to saturate the signal in order to add a kind of grit to it, making it heavier and more aggressive. Fuzz distortions work the same way as plain distortions, but are more aggressive, clipping the signal and turning it into a square wave.

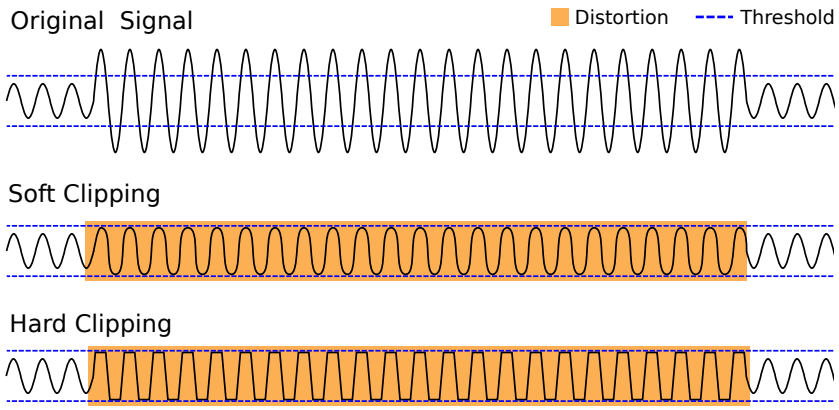


Figure 1: Waveform plot showing different types of clipping

Analogue distortions are created by a circuit-board and electronic components that alter the electrical signal from a guitar, while digital distortions are obtained by some level of software altering a digital representation of the signal. While most guitarists tend to prefer an analogue distortion, the benefit of digital distortions is that they can be added to a clean guitar signal via a computer (although many pedals and amplifiers also use digital distortions). Digital distortions have greatly improved and some virtual amplifiers can now reproduce the sound an analogue device with very high fidelity.

This project uses a digital distortion applied to clean guitar signals via the use of a virtual amplifier on a computer. This does not necessarily simplifies or complicates the task, but it does remove the need of testing the output of the algorithm on real live devices.

2.1.2 Sound Recognition

Human beings are quite accustomed to quickly recognize and classify sounds. Together with other senses, they are used to navigate and respond to the environment. Sounds go even further as they allowed early human beings to develop languages and communicate. The ability to hear sounds led humans to express themselves through the means of music, an arrangement of sounds of different pitches and timbres along a rhythm in order to create a melody. The computer

revolution and the growing interest in data processing and artificial intelligence have inevitably led to the development of sound recognition algorithms. These, however, can take many different shapes and forms depending on the sounds being analysed and the desired goal. Sound recognition involves working with audio signals in order to analyse them and identify patterns, process their data, extract features and classify them. Sound recognition can be used to recognize speech and music, but can also be used for a variety of other applications, such as identifying animal species in areas where it is difficult to visually observe them or in security and monitoring systems, for instance alarms in jewellery stores that recognize the sound of a glass break.

Sound recognition, independently of the type of sound, is usually performed in two steps, feature extraction and classification (Cowling & Sitte, 2003). Feature extraction is the process of extracting characteristics of the sound being analysed, for instance the pitch or the frequency. Classification then compares the extracted features to a catalogue of features of other sounds usually obtained via training. There are two ways to extract features from an audio file, a way that deals directly with the audio signal and a way that uses visual representations of such signal. When using the audio signal directly, Cowling & Sitte (2003) specify that out of the eight commonly used methods in speech recognition, only two of them are commonly used with musical instruments, frequency extraction and mel frequency cepstral coefficients. Visual representations can be produced in different ways, but usually involve representing the values extracted using the methods aforementioned. These are covered in the next section.

2.1.3 Spectral Representations

Feature extraction is an important part of the process of identifying and classifying sounds. Visual representations of an audio signal can help to capture some of these features. Some of them are very efficient and allow for capturing most or all of the features, depending on the sound being analysed.

According to Kim, Moreau and Sikora (2004), a very efficient representation is a representation of the mel-frequency cepstral coefficients (MFCC), obtained

by applying Fourier transform, which allows for separating the different frequencies and represent their magnitude in the original signal. The spectrum is then broken down in several bands representing different frequencies and colours are added to it to represent the magnitude (volume) of each frequency (Fig 2).

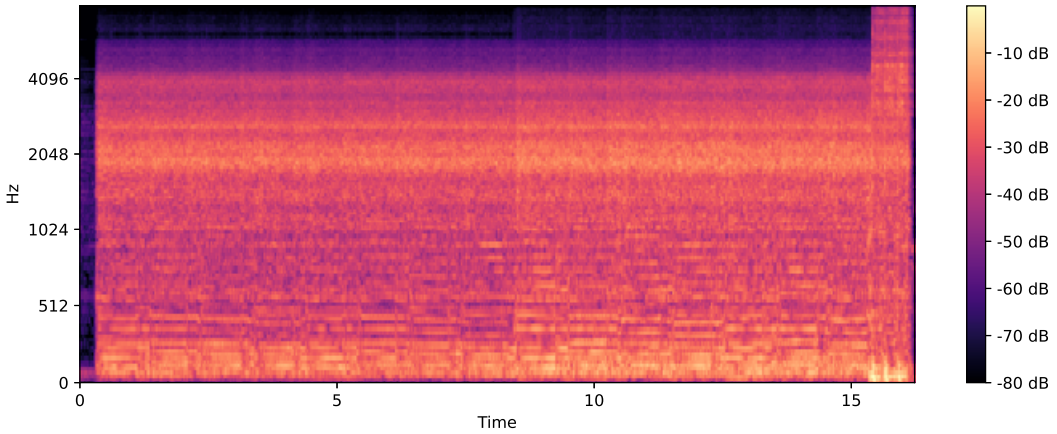


Figure 2: A Mel-Frequency Spectrogram

A representation of the audio spectrum (Fig 3) purely based on frequency is easier to obtain, but is a bit less detail than a mel-frequency spectrogram.

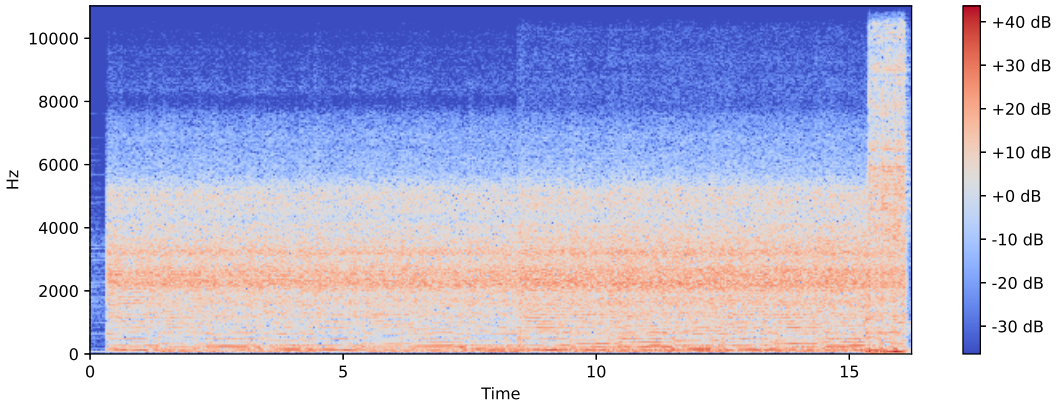


Figure 3: Audio Spectrum Representation

Other representations exist, such as waveform plots, chromagrams or log power spectrograms, but the nature of such representations makes them less adapted for feature extraction.

2.2 Deep Learning

Deep learning is a type of machine learning. LeCun, Bengio and Hintom (2015) explain that for decades, machine learning methods were not very efficient in dealing with natural data in their raw form and required lots of planning and engineering to ensure such data could be represented in a way the algorithm could understand. Deep learning attempts to solve this problem by having the algorithm produce and learn the relevant data representations, known as features, through multiple levels of representations called layers. This section investigates the different techniques commonly used in deep learning.

2.2.1 Neural Networks

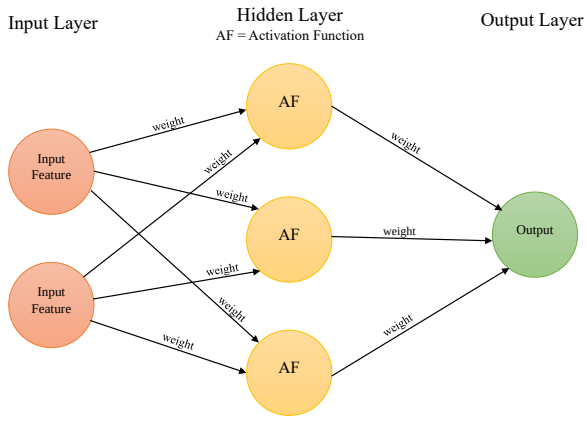


Figure 4: Basic neural network with an input layer, a hidden layer and an output layer

Neural networks are a very common deep learning architecture. As stated by Schmidhuber (2015), a neural network is made of several individual processors called neurons that are connected to each other and run in a specific sequence in order to obtain an output. Some of these neurons receive data directly from their environment (input neurons), while the other neurons receive data from previous neurons. Each neuron runs an activation function which determines

the output of the neuron as a value between 0 and 1 where 1 means that the neuron is activated. An activated neuron equates to an extracted feature. Each output of each neuron is then associated to a specific weight, a number which emphasises or de-emphasises the feature. For instance, a network that aims to recognise that a picture represents an elephant may put more weight on the fact that the image contains a trunk while it may put less weight on the fact that it contains whiskers, a door, or a tree. The sum of each weight is multiplied by the output of the activation function is then passed to the next neuron, which will then process this data with its own activation function, extracting more features. At the end of the process, the found features are compared to an existing library of features and the algorithm returns an output based on this.

2.2.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of neural network that is especially adapted to deal with data that is highly correlated, such as pixels in an image or a video, or words in a textual document (Tian, Ma, Zhang, & Zhan, 2018). This paper focuses on the use of CNN for image classification.

A CNN is made of convolutional layers, pooling layers and fully-connected layers, each with distinct roles.

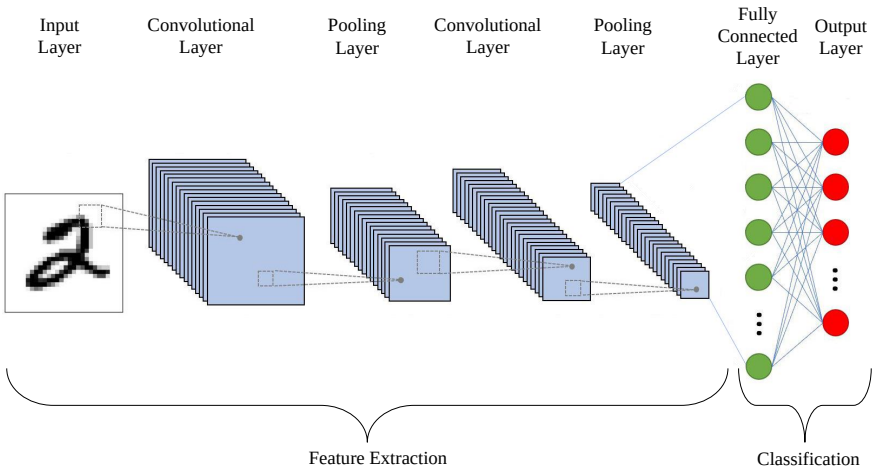


Figure 5: A CNN sequence to classify handwritten digits

A convolutional layer uses filters, which are arrays of pre-determined values, in order to extract features of the input. For instance, a filter that consists in a 3x3 array of values will be applied to each 3x3 blocks of pixels in the image and the inner product of the result will be used to create a new, smaller array of values that represents features. This array of values is called a feature map and can be passed to another convolutional layer in order to extract or refine more complex features. It usually goes through a pooling layer before reaching the next convolutional layer. It is important to note that the values in a convolution filter are the weights that will be trained.

A pooling layer combines multiple values from a previous layer into a single value in order to reduce the size of the array that will be passed to the next layer. As Young, Hazarika, Portia and Cambria (2018) stated, they increase the computation speed and the overall performance of the algorithm.

Finally, fully-connected layers have connections to all nodes in the previous layer and are used to incorporate all previously extracted features into global features with the aim of calculating an output.

2.2.3 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a type of neural network that incorporates the concept of memory, referred to as internal state, which allows for using information from previous inputs to influence both the current input and output (Naz et al., 2015). This makes RNN efficient in identifying patterns in time series dependant data, such as audio and videos.

In other words, a RNN works similarly to other neural networks, but is looped and run several times with a different input each time, and use the information from the previous run to influence or refine the next output. Multiple outputs may be produced, or they may be combined to produce one generic output encompassing all the features extracted from the numerous inputs.

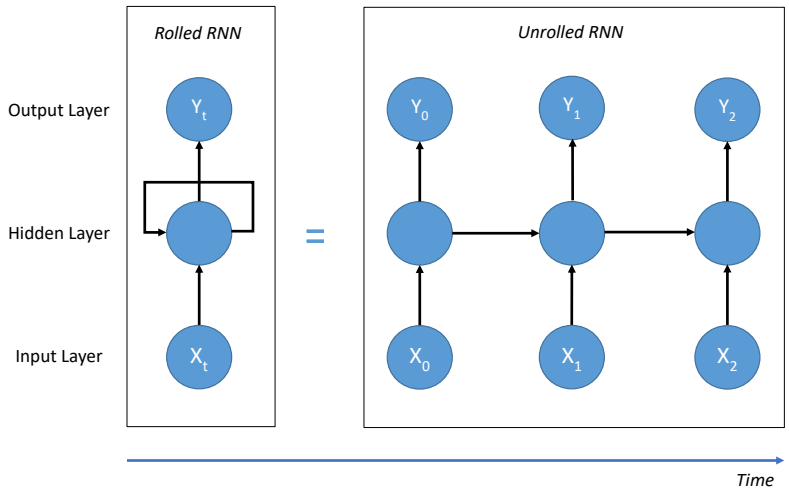


Figure 6: A RNN sequence, rolled (left) and unrolled (right), where we can see the output of the previous run being passed as an input to the next run alongside the next input

A problem that is associated with RNN is the vanishing gradient problem, which occurs when the gradient of the loss function (more on it in the next section) keeps getting smaller each run, eventually becoming so close to zero that the algorithm stops learning (MacDonald, Godbout, Gillcash, & Cairns, 2019). To solve this problem, the most common implementation of RNN is the long short-term memory (LSTM) RNN, which replaces hidden layer units with memory blocks that are controlled by three gates, an input gate, an output gate and a forget gate (Naz et al., 2015). Hu et al. (2020) explain that these gates are respectively responsible for controlling input, output and update of information.

2.2.4 Weights, Gradients, Loss Functions and Backpropagation

When a neural network is given an input, the information travels one way through the network until it produces an output. This is called forward propagation. The output produced by the algorithm is heavily influenced by the weights, which are initialised before the algorithm begins training (Livieris & Pintelas, 2019). There are several ways to initialise weights, for instance:

- Using small random numbers, where the weights are unique numbers very close to zero
- Calibrating the variances with $\frac{1}{\sqrt{n}}$, where n is the number of inputs of a neuron

Since the output mostly depends on the values of the weights, it is therefore paramount to update them according to the current output in order to increase the overall efficiency of the algorithm (Livieris & Pintelas, 2019).

While one can evaluate every prediction and update every weight manually, it turns out that it is possible for the algorithm to perform this action itself. This is achieved by computing the gradient of a loss function at every previous step by flowing backwards through the network (Goodfellow, Bengio, & Courville, 2017). This process is called backpropagation. Once the gradient for a specific neuron has been calculated, another algorithm called stochastic gradient descent is used to update the weight, slightly changing its value, and will do so for every weight in the network. Once all the weights have been updated, the neural network is run once more and the entire process is repeated until the correctness becomes satisfactory. This part constitutes the learning part, or training.

Backpropagation is particularly convenient to use since, as Goodfellow et al. (2017) claim, it allows for evaluating the gradient in a fast, easy to implement and computationally inexpensive way. It also does not requires prior knowledge about the network and tends to work well in most cases.

2.3 Related Work

Deep learning has been used a number of times to emulate and model guitar amplifiers.

Schmitz and Embrechts (2018), as well as Zhang et al. (2018) have proposed using LSTM models to reproduce the non-linear behaviour of a tube amplifier. Their approach consists in using a guitar signal composed of two playing techniques, single notes and chords (multiple notes played at the same time). They interestingly found out that a large dataset is not necessarily needed to achieve

a satisfying result. Schmitz and Embrechts (2018) explain that a twenty seconds dataset was enough for them successfully emulate the sound of a tube amplifier. The produced emulation was reported having less than 1% of root mean square error between the prediction and the actual signal from the amplifier. Zhang et al. (2018), however, reported that their results are not perfectly mimicking the sound of the amplifier, although the signals were very similar.

Gillepsie and Ellis (2013) took a different approach since they created a digital representation of the electronic circuit of a common-cathode tube amplifier and used kernel regression to predict the circuit's states and its output. While they seem to have obtained satisfying results, they also mention that such process comes with a high computational cost.

Wright, Damskägg, Juvela and Välimäki (2020) used both a feedforward series of convolutional layers and a RNN. Their input dataset consisted in several audio files from different amplifiers and each sample was processed 10 times with a different gain value each time (from 1 to 10). The results obtained were very convincing, test subjects not being able to hear a difference between the output samples and the reference samples. They do mention that the RNN implementation was less computationally expensive than the feedforward series of convolutional layers, which is likely to make it the better choice since the outputs were virtually similar.

2.4 Conclusion

The great number of options available in the field of deep learning makes it an interesting project. Sound classification has been rather successful when using neural networks and it seems that computers are powerful in finding differences in images. Similar projects tend to focus on recreating one specific guitar sound without taking care of the settings of the source amplifier. This project, however, aims to help reproduce several guitar sounds from the same amplifier with different settings by returning the settings needed to obtain the same sound. It also aims to use visual representations of the different sounds to be analysed and classified by a CNN, which is a somewhat new approach as most projects tend

to work directly with an audio signal and a RNN. Moreover, this project aims to develop a GUI to make the software easy to use, a feature which seems to be lacking in similar projects.

The choice of using a CNN was made as it is possible to observe differences with the naked eye between visual representations of audio tracks processed with different amplifier settings. If a human being can see a difference between those files, it is likely that a computer should be able to perform at least as well as a human, if not better. This project will investigate whether classification could be achieved by the means of visual representations of audio samples rather than working directly with the audio. Although this may have been attempted in other fields, such as biology or marine life, it is quite novel in terms of guitar sounds.

This project intends to help guitarists to train the algorithm with their own amplifier in order to to be able to retrieve or keep track of the settings they like. It could also be trained on other amplifiers to provide more flexibility, for instance a guitarist playing in a band could train it to retrieve the settings of other band members' amplifiers which could be useful in the future if they wished to reproduce a sound they liked and have perhaps forgotten how to achieve it.

It also leaves room for improvement as such software could possibly be upgraded to reproduce the sound of the source amplifier.

3 Methodology and Implementation

The project involves creating a dataset of mel-spectrograms to be used to train a deep learning CNN model to recognize and classify audio files, or in this case the visual representation of such audio files, based on the amplifier settings that were applied to it. The project also requires a GUI to be developed to allow for a user to easily use the classifier and compare the results given by the classifier to pre-recorded samples.

This section covers the pre-development which explains how the audio files were obtained and the virtual amplifier used. It also covers how the dataset was created and what the mel-spectrograms look like and represent, before moving on to the actual CNN implementation, training, testing, and performance review. Finally, it explains how the GUI was developed and tested and provides an overview of the user experience.

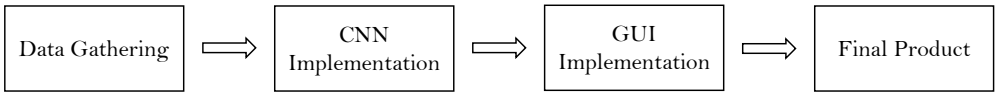


Figure 7: Methodology diagram

3.1 Pre-Development

3.1.1 Practice

The data that was used to train the algorithm had to be produced first, since it does not exist or is not publicly available.

To add distortions by the means of a virtual amplifier requires direct-input (DI) guitar tracks. A DI track is a track that records the sound straight out of the pick-ups of the guitar. Such tracks are especially useful when working with virtual amplifiers.

In order to record the DI tracks, an electric guitar was plugged into an audio interface and the tracks were recorded with Ardour, an open-source digital audio

workstation (DAW). Amplifier effects were subsequently added to each of the tracks.

Figure 8: Audio interface used as DI Box (left) and Ardour (right)

The distortions were added using the virtual amplifier Kuassa Amplifikation Lite. This choice was made because it is a free virtual amplifier that can be used in any DAW. This choice makes the reproduction of the experience easy.

Figure 9: A virtual amplifier (Kuassa Amplifikation Lite)

One hundred and twenty DI guitar tracks have been recorded. Each of the track lasts 5 seconds. The algorithm currently classifies audio files in twelve categories or classes. After processing each track with the amplifiers effects, it resulted in a 2.04 Gb dataset of 1440 wave tracks.

3.1.2 Terminology

To understand how the dataset was created and what each of the files represent, one must understand the different settings on the amplifier.

In order to reduce the amount of training data needed, some of the settings in the amplifier were not taken into account and have the same value for each and every one of the files in the dataset. These settings are the input gain, the gain, the presence, the volume and the cabinet. Because these settings remain unchanged, they do not not affect the mel-spectrograms nor the training.

- The input gain is responsible for calibrating the volume of the audio file before it is being processed by the virtual amplifier.
- The gain is responsible for the amount of audio data that will be processed by the amplifier. A higher gain usually means a greater distortion. It determines how strong the signal should be at the start of the processing chain.
- The presence knob works similarly to the treble knob (more on it in the next paragraph), but instead of being able to decrease or increase the amount of high frequencies in the audio file, the presence knob can only slightly increase them in order to make them feel more 'present'. It is a very common setting in amplifiers that brings a kind of brightness to a distorted guitar, especially when playing melodies (e.g. lead parts, guitar solos), where the lower frequencies can make the sound 'muddy'.
- The volume knob controls the volume of the output at the end of the processing chain. Distortion tends to increase the overall volume so the volume knob may be used to hear the distortion at the volume chosen by the user (e.g. decrease it when adding a heavy distortion or increase it when going for the clean channel).
- The cabinet represents the 'set of speakers' used by the amplifier to produce sound. Physical amplifiers come in two ways, as combo amplifiers where the effect processing part (known as the head) and the speakers (the cabinet) come in one product, or as the head and cabinet separated. This virtual amplifier offers three settings, internal cabinet (the virtual amplifier is responsible for producing its own audio), off (no sound), or external where

it is possible to connect an external cabinet plugin. For this experience, the internal cabinet was used.

The table below shows the value of these settings (that remained unchanged throughout the experience).

Input Gain	**Gain**	**Presence**	**Volume**	**Cabinet**
2.5	5	5	5	Internal

Table 1: The amplifier settings that remained unchanged

The settings that were changed and that the deep learning algorithm attempts to recognize and classify are the amplifier channel, and the equalizer settings bass, middle and treble.

- The amplifier channels are dedicated channels for a specific distortion. Typically, one of the channels is devoted to clean tones and usually features a built-in compressor to address the volume loss that is common with clean tones, while the other channels are devoted to different types of distortions that could go from 'bluesy' to 'heavy'. In the case of the used amplifier, there are three channels. Clean, which allows for clean tone (but can become bluesy if the gain is turned up), rhythm, which is a strong distortion and lead which is a heavy distortion. The channel chosen greatly affects the sound.

- The bass knob increases or decreases the range of lower frequencies to the user's liking. It simply works by adding or removing some gain only on the portion of the audio signal that falls in that frequency range.

- The middle knob similarly adjusts gain on mid-range frequencies.

- The treble knob similarly adjusts gain on higher range frequencies.

More details on the settings used to train the algorithm can be found in the next section.

3.2 Convolution Neural Network

3.2.1 Dataset

The aim of the project being to train a CNN, a mel-spectrogram has been produced for each of the tracks using Librosa, a Python library designed for audio and music processing. Amongst the many features, Librosa incorporates feature extraction tools in the form of different visual representations of audio signals. The mel-spectrogram representation was chosen as it appeared to be the visualisation in which a greater difference could be observed between the different sounds.

The dataset that is used for training, validation and testing is therefore a lighter dataset of 120 Mb containing the 1440 mel-spectrograms in PNG format. Table 2 shows the 12 classes and Figure 10 shows the difference between mel-spectrograms for guitar sounds with different amplifier channels.

Class	**Amplifier Channel**	**Bass**	**Middle**	**Treble**
1	Clean	0	0	10
2	Clean	0	10	0
3	Clean	10	0	0
4	Clean	5	5	5
5	Rhythm	0	0	10
6	Rhythm	0	10	0
7	Rhythm	10	0	0
8	Rhythm	5	5	5
9	Lead	0	0	10
10	Lead	0	10	0
11	Lead	10	0	0
12	Lead	5	5	5

Table 2: The 12 classes the algorithm will be trained with

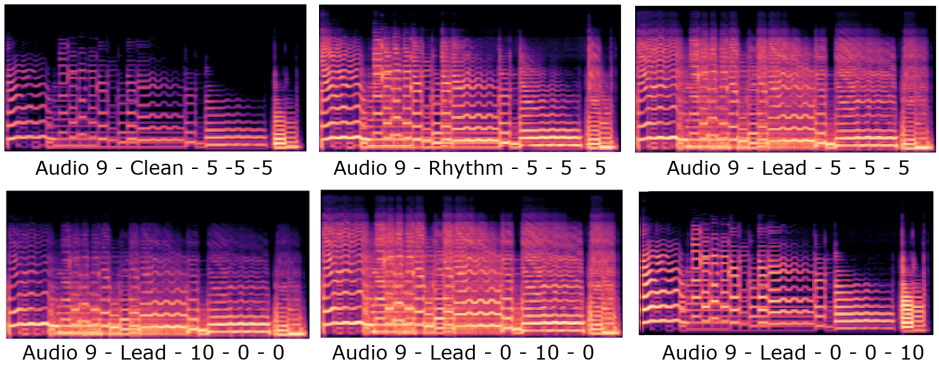


Figure 10: Mel-spectrograms of the same audio file with 6 different amplifier settings (e.g. Classes)

Multiple datasets were created depending on the implementation chosen. All datasets were separated in training, validation and testing groups. They were typically split by using 60% of the data for training, 20% for validation and 20% for testing.

The folder structure was built as per the keras CNN implementation requirements. As such, three folders called 'train', 'validate' and 'test' have been created. The test folders contains files from all twelve classes mixed together, while the train and validation folders contain files sorted in subfolders according to their own category. Figure 11 shows the structure used to train the binary classifiers and Figure 12 shows the structure used to train the multi-class model.

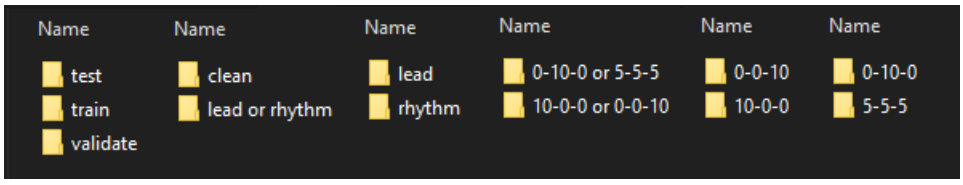


Figure 11: Dataset structure used to train the binary models (all models contain the test, train and validate folders, each model has a different set of classes

Figure 12: Dataset structure used to train the multi-class model, whole structure (left) and inside the validate and train folders (right)

3.2.2 CNN Implementation

The CNN has been implemented using the built-in Keras Sequential model. The sequential model is a linear model that implements a linear stack of layers. This means that between the input and the output, the defined layers will be travelled through linearly, in the order they were added.

The retained model makes use of a total of 9 layers. The input layer is a Conv2D layer, that also specifies the input shape. The input shape does not need to be specified in the next layers as they perform automatic shape inference. This layer is followed by a MaxPool2D pooling layer. The next 4 layers are 2 sets of Conv2D and MaxPool2D layers. Each Conv2D layer sees its number of filters doubled in order to extract more features as the patterns get (or should get) more complex. The model continues with a Flatten layer, that reduces the dimensionality so its output can be passed to the next two Dense layers. Dense layers are layers of neurons, as opposed to convolutional (Conv2D) and pooling (MaxPool2D) layers. The first Dense layer takes the role of the fully connected layer and the last Dense layer is the output layer. The batch size represents the number of images that are used during each step per epoch, where an epoch

represents one pass over the entire dataset. The final dataset containing 1440 images, a batch size of 32 was chosen as it is allows for going through the entire dataset in 45 steps per epoch.

The model parameters are defined as per table 3.

Layer	Parameter	Value
	Batch Size	32
Conv2D (input layer)	Number of filters	16
	Kernel Size	3x3
	Activation	Relu
	Input shape	135x270
MaxPool2D	Pool Size	2x2
Conv2D	Number of filters	32
	Kernel Size	3x3
	Activation	Relu
MaxPool2D	Pool Size	2x2
Conv2D	Number of filters	64
	Kernel Size	3x3
	Activation	Relu
MaxPool2D	Pool Size	2x2
Flatten	No parameter	
Dense	Number of units (Neurons)	256
	Activation	Relu
Dense	Number of units (Neurons)	1
	Activation	Sigmoid

Table 3: Model parameters

Depending on the problem at hand (multi-class or binary), the class mode and the loss function were different. When training the multi-class model, the class mode was *'categorical'* and the loss function *'categorical crossentropy'*. When training the binary class model, the class mode was *'binary'* and the loss

function *'binary crossentropy'*. The batch size and number of steps per epoch were different when training different binary classifiers. The first two binary classifiers (amplifier channel) were trained with a batch size of 20 and 48 steps per epoch. The last three binary classifiers (equalizer settings) were trained with a batch size of 32 and 45 steps per epoch. The difference is due to the fact that the first two models were trained with less data to make the computation faster. The results were not affected.

3.2.3 Training

Several experiments were performed with different number of epochs, steps per epochs and batch sizes. These are covered in detail in the evaluation and results section of this report.

Generally, training the multi-class model took longer than training binary models individually. The compile time and the accuracy were generally better when training binary models.

One issue that arose from training the algorithm is that is appears the amount data available is not sufficient for a multi-class problem. Good accuracy was obtained during training, but the accuracy on test data (data that is neither part of the training set nor the validation set) tended to be lower. For this reason, multiple trainings have been performed on sets of 2 classes. The end result is transparent to the user, the audio file will be classified in one of the 12 classes, but the process to reach this outcome actually makes use of several trained models. Each of these models were trained on two classes only, for the reason that the accuracy obtained is much higher.

For instance, when training the model to classify files per amplifier channel, the classifier generally failed classifying 'Rhythm' and 'Lead' channels, while it would always get the 'Clean' channel right. When training the model only with the classes 'Rhythm' and 'Lead', the model performed very well with an accuracy of 97% on test data. The amp channel classification is therefore performed in two times, first classifying between 'Clean' and 'Rhythm or Lead', and if the file falls into the second category, another model performs the classification between

'Rhythm' and 'Lead'.

This process has been repeated as many times as necessary to achieve a decent and accurate classification. Figure 13 shows the classification flow of the multiple binary classifiers.

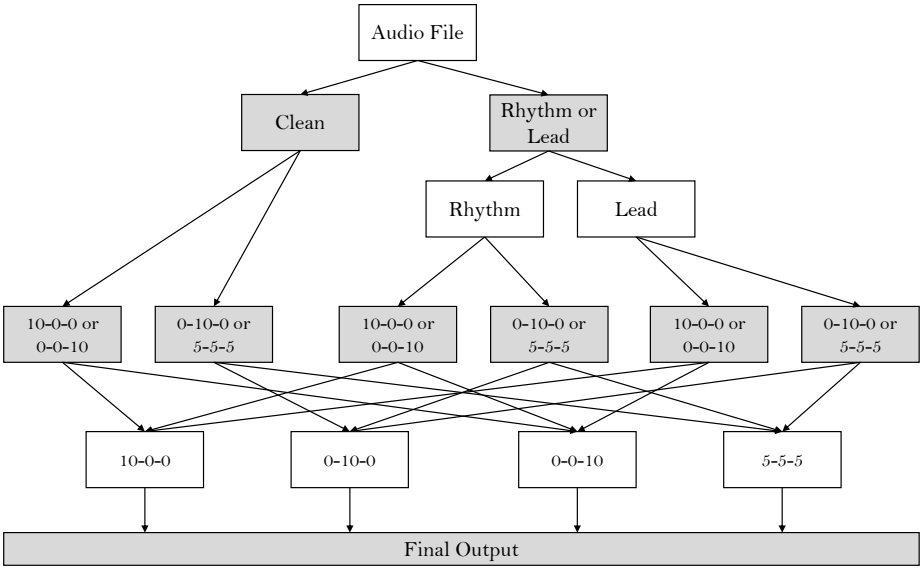


Figure 13: The multiple classifications performed to reach the final output

Five binary models were trained and one multi-class model was trained. They were all retained and implemented in the GUI, so the user can run the classifier of his choice and decide which one performs better.

3.2.4 Testing

Each model was carefully tested on test data. Test data is data that is neither part of the training set nor the validation set, and therefore provides an overview about how well the model performs on new data.

The testing was performed in Python by loading all the files in a test folder in array and asking the model to predict their class. The output was printed in a table along with statistics such as how many predictions were correct and how many were incorrect. This was achieved by means of homemade methods.

The figure below shows a short sample of such output. The complete outputs

for all models can be found in Appendix E.

```
+----------------------------+-------------------+------------+----------+
| File                       | Class             | Prediction | Result   |
+----------------------------+-------------------+------------+----------+
| 2 Minutes To Midnight 2    | clean-0-10-0      | 0-10-0     | PASS     |
| 2 Minutes To Midnight      | clean-0-10-0      | 0-10-0     | PASS     |
| 8 Strings Low 2            | clean-0-10-0      | 0-10-0     | PASS     |
| Chords 1 -2                | rhythm-5-5-5      | 5-5-5      | PASS     |
| Chords 2 -2                | rhythm-5-5-5      | 0-10-0     | FAIL     |
| Chords 3 -2                | rhythm-5-5-5      | 5-5-5      | PASS     |
| Chords 1                   | rhythm-5-5-5      | 5-5-5      | PASS     |
+----------------------------+-------------------+------------+----------+

Pass: 98.33% (118), Fail: 1.67% (2), Total: 120
+---------+-----------+--------+---------+
| Classes | Precision | Recall | F-Score |
+---------+-----------+--------+---------+
|  0-10-0 |    97.0   | 100.0  |  0.98   |
|  5-5-5  |   100.0   |  97.0  |  0.98   |
+---------+-----------+--------+---------+
```

Figure 14: Sample model output on test files

The output provides four metrics: accuracy, precision, recall and f-score. These are covered in detail in the results part of this report.

3.2.5 Final Scripts

Once the models were trained, simple python scripts have been created in order to load the trained models and perform the classification that will be returned to the user.

The take the form of two files:

- `single-pred-multi.py`
- `single-pred-binary.py`

The files take as argument the path to the mel-spectrogram that needs to be classified. The output is a simple string that returns the predicted class.

3.3 Graphical User Interface

3.3.1 GUI Implementation

The Graphical User Interface (GUI) is developed using the .NET Core Windows Presentation Foundation (WPF). WPF is a user interface framework that uses a vector-based rendering engine. The visual aspects of the software are defined using XAML files, similarly to Android and iOS native applications, and differently from older desktop interfaces such as Windows Forms. It is a more modern approach to both the UI design and the general design as WPF strongly suggests the use of the Model-View-ViewModel (MVVM) pattern. MVVM has been used to develop this product. Indeed, the visual aspect is defined in the XAML file (View), a specific class is responsible for handling the controls such as clicks and inputs from the user (ViewModel), and several other classes are responsible for the logical processing of the information (Model).

This makes the whole process extremely simple and allows for keeping clarity. If any changes were required in the future, it would be very easy for any developer with knowledge of MVVM to bring changes to the application. The logical operations are performed in C#.

The GUI is defined in the file MainWindow.xaml, while the controls are handled by the rather short class MainWindow.cs. The entirety of the processing is performed by multiple classes that perform the required task and return an output to the user. As such, the class SoundProcessing.cs is responsible for loading the audio file and its information, as well as building the audio player and the waveform renderer (see Fig. 15). The class PythonScripts.cs is responsible for running the Python scripts and obtain an output from them (see Fig. 16). Finally, the class CompareResults.cs is responsible for loading the audio samples that are used in the "Comapre Results" page of the program (see Fig. 17).

This allows for the code to be kept clean, nicely separated and therefore allows for a programmer to easily understand and amend any part of the software as required.

3.3.2 Libraries

NAudio is the only external library that is used in this project. It is used to load the audio and build the audio player. The audio player was developed from scratch, NAudio only provides means to play audio. The capabilities of NAudio are greater than the default built-in C# SoundPlayer. NAudio loads the audio files as an array of 32-bit integers and provides functionalities to keep track of the position of the reader in the array, in other words it allows for keeping track of time. This is a very useful feature for building an audio player. NAudio also includes another open source library called WaveFormRenderer that allows for creating a waveform (i.e. a graph that represents the amplitude of a sound over time) that was particularly useful to make the audio player look nice and professional.

3.3.3 Background Workers

Since the deep learning model was trained using Python and the keras/tensorflow library, the GUI needs to run Python scripts in order to provide a result to the user.

While the option of using IronPython (which is a kind of python virtual environment that lives within the C# application) was considered, it was not retained as a viable solution as it only works with Python 2. Instead, a virtual Python 3 virtual environment was created and the relevant libraries were installed on it. The python files that need to be run therefore connect to this virtual environment and perform the task.

In order to prevent blocking the UI thread and provide a seamless user experience, the Python files are accessed asynchronously using background workers. This means that each of the python scripts runs in the background and return an output whenever the process is finished.

This design choice works very well and performs very fast. A challenge that arises, however, is the need for the Python virtual environment to be present in the application folder. This requires an installer to uncompress and copy the

files in the installation directory chosen by the user, just like any traditional installation process. This is not a big problem since quite a few external files are needed for the program to work, such as external libraries and some image resources, which means that an installer would be required in any case. A problem, however, could be the size of the virtual python environment. Indeed, the virtual environment requires a number of libraries, such as numpy, matplotlib or librosa. These libraries are rather small in size, but the environment also requires tensorflow/keras to be installed in full. Tensorflow alone weighs around 320 Mb, meaning that the program, no matter how small and simple it may be, will be at least as heavy as the Python virtual environment. This is covered in details in the evaluation section.

Considering the heavy weight of the whole virtual environment (around 1Gb), it has been chosen to remove it from the final product. Instead the user is asked to provide a path to his local 'python.exe' file. It is, of course, necessary that all required packages are installed (see Appendix F).

3.3.4 Flow

The GUI is made of three pages. Page two and three are disabled upon software start. The first page requires the user to load an audio file. Any audio file can be loaded, but the AI classification will only work with a wave file. It is possible however to load another audio format and listen to it. Once an audio file has been loaded, the next page becomes enabled. On this page the user can see the mel-spectrogram that was produced and run the classifier. Once the classifier has run, the last page is enabled which allows the user to compare his loaded sample to actual samples recorded with the same amplifier and different settings.

3.3.5 User experience

The graphical user interface works in three steps.

The first step offers a user the option to load a wav file and view its basic information as well as the option to listen to the track. These options allow for

the user to ensure the right file in the right format has been loaded.

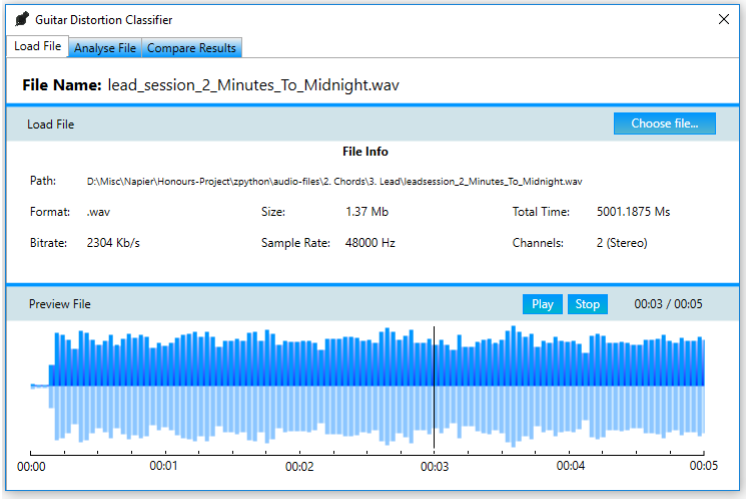


Figure 15: The first window upon opening the GUI where wav files are loaded

The second step allows the user to see the mel-spectrogram that was produced and to get a classification from the trained model.

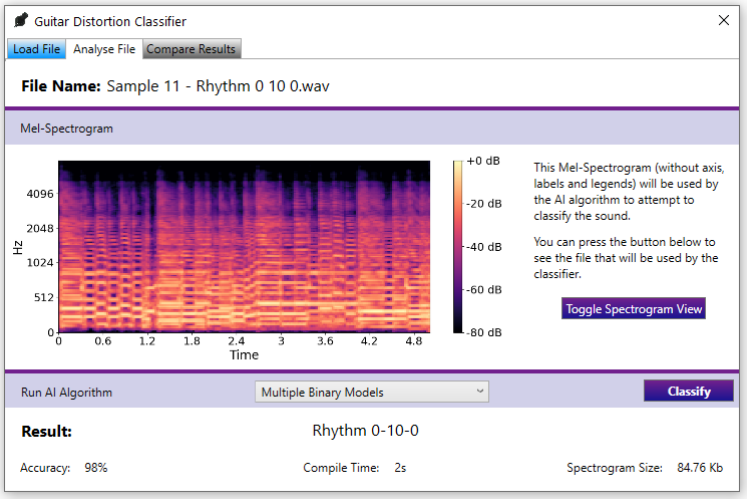


Figure 16: The second window allows for a user to run the classifier

Finally, the last step allows the user to compare the classification that was returned to him by the means of audio samples provided by the software.

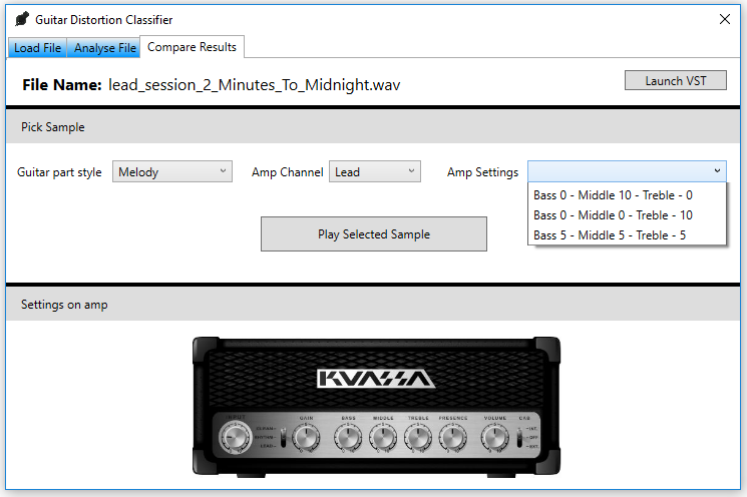


Figure 17: The third window allows for comparing the different amplifier settings

3.3.6 Testing

Extensive testing has been carried out to ensure the software is fit to use.

Test Case	**Action**	**Expected Output**	**Actual Output**	**Pass/ Fail**
1	Load a wave file	Wave file loaded, mel-spectrogram produced and 'Analyse tab' enabled	As expected	Pass
2	Start wave file playback	Wave file plays, time displayed by audio player is correct	As Expected	Pass
3	Load a mp3 file	mp3 file loaded, error message displayed and 'analyse tab' not enabled	As expected	Pass

Continue on the next page

Test Case	**Action**	**Expected Output**	**Actual Output**	**Pass/ Fail**
4	Start mp3 file playback	mp3 file plays, time displayed by audio player is correct	As Expected	Pass
5	Display file picker and ensure only audio files can be selected	only .wav, .aac, .aac and .mp3 files are shown	As expected	Pass
6	View 'analyse' tab	Mel-spectrogram of loaded wav file is displayed	As Expected	Pass
7	Run the classifier	Waiting message displayed then replaced by output, 'Compare results' tab enabled	As expected	Pass
8	Select and play sample sounds on 'Compare results' tab	Sounds play correctly based on the user's selection	As Expected	Pass
9	Pick a new file after one has already been loaded	New file loads and 'analyse' and 'compare results' tab disabled (if .wav file, 'analyse' tab enabled)	As Expected	Pass

Table 4: Test cases

4 Evaluation and Results

This section provides a detailed explanation of the experiments that were conducted in order to choose the final model along with their performance review. It also provides a detailed overview of the final results.

4.1 Experiments

4.1.1 Metrics

To understand how the following results were evaluated, a number of metrics are used.

During training, these metrics are:

- Loss: this was covered in detail in part 2.2.4 of this report. The training loss is compared to the validation loss. If the validation loss is greater than the training loss, this may indicate overfitting (depending on how much and how often the validation loss is greater). If the training loss is greater than the validation loss, this may indicate underfitting. Ideally, the model should stop training when the two of them converge to 0, in order to prevent overfitting (the whole backpropagation mechanism aims to minimize the loss function).

- Accuracy: The training accuracy is compared to the validation accuracy. The fact that the training accuracy becomes better may only indicate that the model learned how to classify the data in the training set, but does not necessarily tell us how well the model performs with new data. The validation accuracy therefore shows how the model performs with the validation set, data that is different from the one it is being trained on. Ideally, the model should stop training when the two of them converge to 1 (accuracy is measured between 0 and 1 i.e. 0% and 100%).

During testing, the metrics used are:

- Accuracy: Percentage of correctly predicted elements over the total number of elements.

- Precision: Percentage of correctly predicted elements in a specific class over the total number of elements predicted in this specific class
- Recall: Percentage of correctly predicted elements in a specific class over the total number of elements belonging to this specific class
- F-Score: Harmonic average of precision and recall calculated as $2 \times \frac{recall \times precision}{recall + precision}$

4.1.2 Multi-class

As mentioned earlier, the objective is to classify the mel-spectrograms in twelve categories. The first experiment consisted in training a model with the twelve classes right away. The training took a long time, most likely because the whole dataset is much larger than the datasets used in binary class training.

Good results were obtained, with training and validation accuracies of 97%. However, when testing the model on test data, only 80% of the files were classified correctly.

Figure 18 shows the loss values converging to 0 while the accuracies converged to 1.

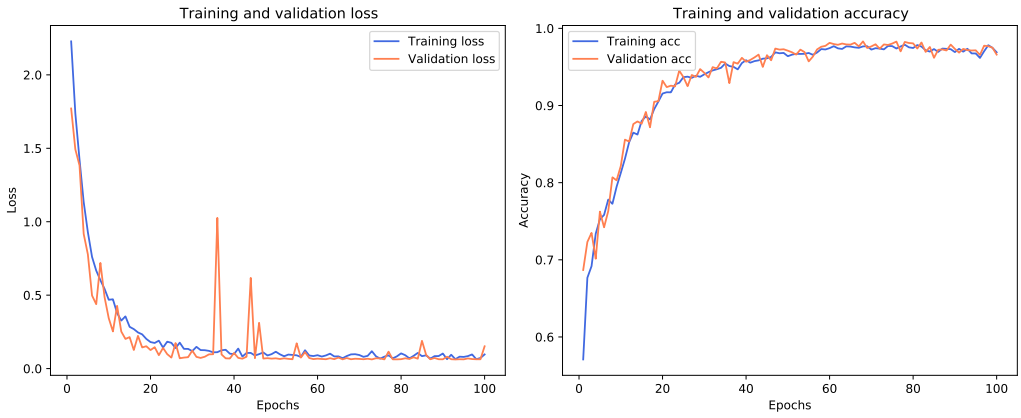


Figure 18: Training and validation loss (left) and accuracy (right) when training the model with 12 classes

While it is generally good, some of the classes showed problems. One of them

was being predicted wrong 100% of the time and some, such as ”clean-0-0-10”, are being predicted more often that they should. Figure 19 shows the overall performance of the multi-class model.

```
Pass: 81.67% (98), Fail: 18.33% (22), Total: 120
+---------------+-----------+--------+---------+
|     Class     | Precision | Recall | F-Score |
+---------------+-----------+--------+---------+
|  clean-0-0-10 |    62.0   | 100.0  |   0.77  |
|  clean-0-10-0 |   100.0   | 100.0  |   1.0   |
|  clean-10-0-0 |    77.0   | 100.0  |   0.87  |
|  clean-5-5-5  |   100.0   |  70.0  |   0.82  |
|  lead-0-0-10  |   100.0   |  40.0  |   0.57  |
|  lead-0-10-0  |    77.0   | 100.0  |   0.87  |
|  lead-10-0-0  |   100.0   |  70.0  |   0.82  |
|   lead-5-5-5  |   100.0   | 100.0  |   1.0   |
| rhythm-0-0-10 |   100.0   | 100.0  |   1.0   |
| rhythm-0-10-0 |    50.0   | 100.0  |   0.67  |
| rhythm-10-0-0 |    0.0    |  0.0   |    0    |
|  rhythm-5-5-5 |   100.0   | 100.0  |   1.0   |
+---------------+-----------+--------+---------+
```

Figure 19: Output of the multi-class model on test data

These results must be interpreted. A perfect score is a class having an f-score of 1, meaning and the precision and recall both equal to 100%. In the case of class ”clean-0-0-10”, we observe a recall of 100%, which means that all instances of this class were classified correctly. The precision, however, is only 62%. This means that this class was also predicted on items that did not belong to it. The f-score therefore shows that the overall accuracy for this specific class is around 77%. On the other hand, the class ’clean-5-5-5’ shows a precision of 100% and a recall of 70%. In this case, 7 and only these 7 of the 10 elements belonging to this class were predicted as belonging to it. This is why the precision reached a level of 100% (7 out of 7 are correct, no other element was predicted to belong to this class). The recall, however, includes the 10 elements that should have been predicted (7 out of 10), hence the 70% score. It is important to observe that the class ’rhythm-10-0-0’ was systematically predicted wrong. Classes that reached a 100% score in both precision and recall were predicted correctly and no other element that did not belong to this class were predicted as belonging to it.

To train the multi-class model, it was attempted to use data augmentation. Data augmentation is a method that slightly modifies images in the datasets in order to create more of them and eventually increase the effectiveness of the algorithm. Data augmentation works by shifting, rotating, zooming or cropping the existing images.

While the accuracy on training and validation data was similar to what can be observed in figure 18 above, the accuracy on test data turned out being disastrous. The algorithm classified a mere 41% of the test data correctly. It is by far the worst outcome of all experiments. Data augmentation was abandoned after a few trials. It was found not suitable for the current problem.

It was found that the reason for this poor performance is due to the model being trained on graphs (mel-spectrograms). Indeed, when a user feeds an input to the classifier, the mel-spectrogram will always be ordered in the same way, on x and y axis, in a png file of the same dimensions. Data augmentation therefore tricks the algorithm into believing different inputs may be provided. Since this is not the case, the overall performance suffers.

4.1.3 Binary Classifiers

Another experiment was to train multiple models that are binary models, which means that they can only classify elements in the dataset in two categories.

In order to obtain the same final output as the multi-class model, five models were trained. The first one was trained to classify the amplifier channels between ”clean” and ”rhythm or lead” and the second model between ”rhythm” or ”lead”. The next model classifies the samples based on their equalizer settings in two categories ”10-0-0 or 0-0-10” and ”0-10-0 or 5-5-5”. Finally, the last two models are used to perform a final classification between ”10-0-0” and ”0-0-10” or ”0-10-0” and ”5-5-5”, based on the previous output.

Figures 20 and 21 show the loss and accuracy values for 2 of such models.

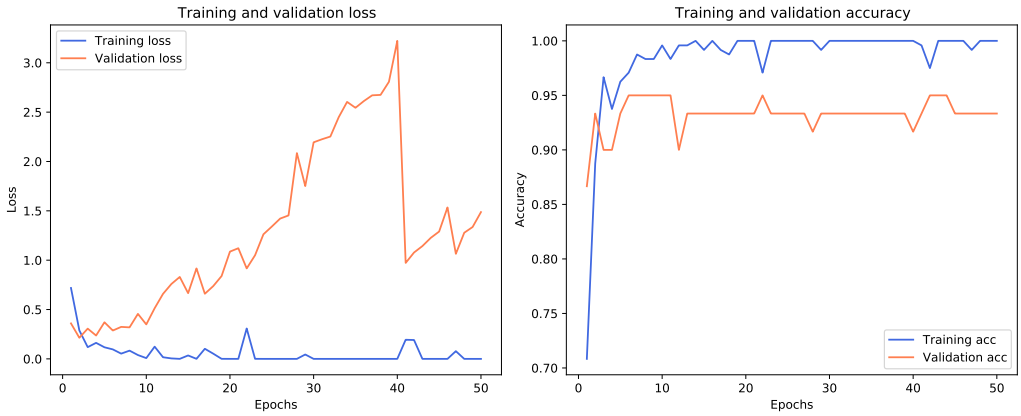


Figure 20: Training and validation loss (left) and accuracy (right) when training the model with 2 classes ”clean” and ”rhythm or lead”

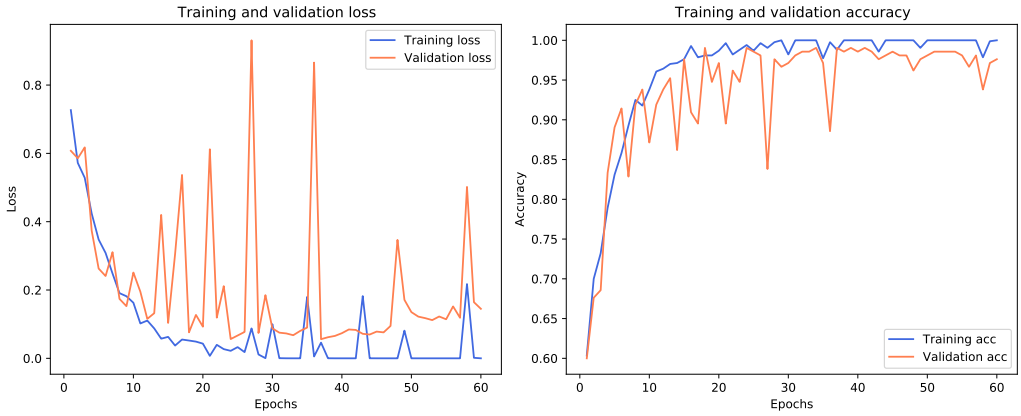


Figure 21: Training and validation loss (left) and accuracy (right) when training the model with 2 classes ”10-0-0 or 0-0-10” and ”0-10-0 or 5-5-5”

These two figures show a very different performance when compared to the multi-class model. Figure 20 clearly shows some overfitting and that the validation accuracy (93% on average) is lower than the training accuracy. Figure 21 shows a a result closer to the multi-class model. The main difference, however, occurs when running these models on test data. In the first case (Figure 20), the model was able to classify correctly 100% of the test data and in the second case (Figure 21) classified 98% of the test data correctly.

```
Pass: 100.0% (60), Fail: 0.0% (0), Total: 60
+----------------+-----------+--------+---------+
|     Class      | Precision | Recall | F-Score |
+----------------+-----------+--------+---------+
|     clean      |   100.0   | 100.0  |   1.0   |
| lead or rhythm |   100.0   | 100.0  |   1.0   |
+----------------+-----------+--------+---------+
```

Figure 22: Output of 'clean' vs 'lead or rhythm' on test data

```
Pass: 98.56% (206), Fail: 1.44% (3), Total: 209
+------------------+-----------+--------+---------+
|      Class       | Precision | Recall | F-Score |
+------------------+-----------+--------+---------+
| 0-10-0 or 5-5-5  |    99.0   |  98.0  |   0.98  |
| 10-0-0 or 0-0-10 |    98.0   |  99.0  |   0.98  |
+------------------+-----------+--------+---------+
```

Figure 23: Output of '10-0-0 or 0-0-10' vs '0-10-0 or 5-5-5' on test data

The performance of these models in their assigned scope lead to explore and experiment a classification with multiple models rather than focusing on one multi-class model.

Other models, such as the one that can be seen in Figure 24, reached 100% accuracy both in training and validation accuracy, as well as when running on test data.

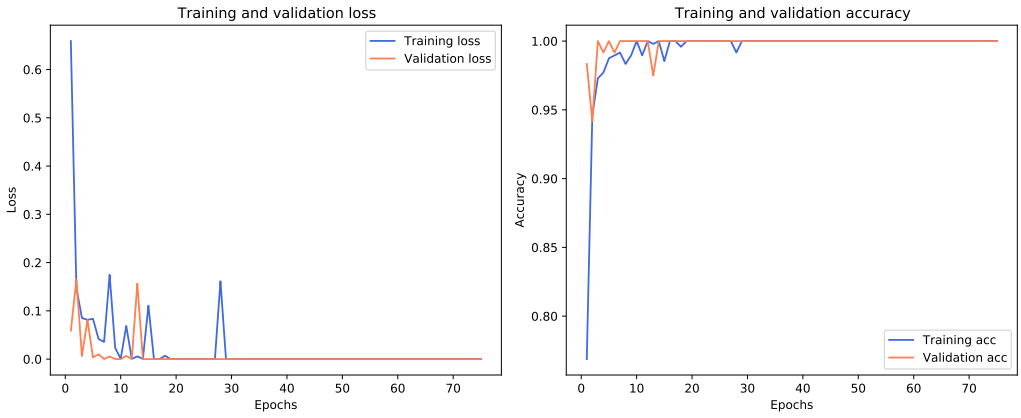


Figure 24: Training and validation loss (left) and accuracy (right) when training the model with 2 classes "10-0-0" and "0-0-10"

All of the performance graphs and the test outputs can be found in Appendix E.

4.2 Results

This section provides an overview of the final results and their estimated accuracy when classifying new data. The table below shows the results obtained on test data for each model.

Model Name	**Classes**	**Type**	**Accuracy**	**Fail**
12-classes.h5	All 12 Classes	Multi-Class	81%	19%
CLR.h5	'clean' vs 'lead or rhythm'	Binary Class	100%	0%
LR.h5	'rhythm' vs 'lead'	Binary Class	98%	2%
EQ1.h5	'1000 or 0010' vs '0100 or 555'	Binary Class	99%	1%
EQ2.h5	'1000' vs '0010'	Binary Class	100%	0%
EQ3.h5	'0100' vs '555'	Binary Class	98%	2%

Table 5: Accuracy of each model on test data

When using the binary models, the final accuracy depends on the route that was taken to reach the final output. The table below shows the average accuracy for every possible series of models.

Route	**Accuracy**	**Fail**
CLR.h5 → EQ1.h5 → EQ2.h5	99.33 %	0.66%
CLR.h5 → EQ1.h5 → EQ3.h5	98.66 %	1.33%
CLR.h5 → LR.h5 → EQ1.h5 → EQ2.h5	99.25% %	0.75%
CLR.h5 → LR.h5 → EQ1.h5 → EQ3.h5	98.75% %	1.25%

Table 6: Average accuracy of all models in a specific model series

Finally, the table below provides the final average accuracy of all routes in the above table compared to the accuracy of the multi-class model.

Prediction Method	**Accuracy**
Multi-class model	81%
Multiple binary models	98.5%

Table 7: Accuracy of the two prediction methods

4.3 Discussion

The experiments and the results have shown that it is indeed possible to train a convolutional neural network to classify electric guitar distortions. While the results varied based on the different model parameters used, it is possible to train a model to be highly efficient in recognising and classifying guitar sounds.

If having enough time, a powerful enough machine and enough data, it seems to be a better approach to train only one model, as classifying new data will be much faster. Indeed, only one model needs to be loaded in memory. This however requires more data, and gathering data is one of the difficult aspects of this project.

The other approach, which essentially is a cascade of binary classifiers, has yielded very good results. However, as some of the amplifiers settings were not taken into account, this method seems problematic as the problem will grow exponentially in the future. Adding one more setting into account, such as the input gain, could result in twice as many classifiers (more input gain may trick the current models to classify rhythm parts as lead parts). This makes the multi-class model a better approach as the number of categories can be expanded indefinitely.

The models were trained using 5 seconds long samples. An attempt to classify a file that is more than 5 seconds long may or may not give good results as the mel-spectrograms produced by the algorithm always have the same size. To display more data in the same space may lead the algorithm to classify the sample wrongly. To ensure consistency, a mechanism that selects a 5 seconds sample of a longer file is required.

While it is possible to use a CNN to perform such classification, it appears

that the great variety of combinations of amplifier settings would make it a challenging task to predict the value of each and everyone setting accurately. At this stage, a CNN is a viable option to get a good idea of the overall settings, but further adjustment may be needed by the user to obtain the same sound.

5 Conclusion

5.1 Critical Evaluation

An extensive review of the available literature has been carried out and allowed for gaining a better understanding of what could realistically be achieved. The implementation, however, revealed a number of drawbacks and challenges. One the most difficult task was to gather the required data. The project lead to the recording of a hundred and twenty audio tracks and every one of them had to be processed and exported twelve times with different amplifier settings. Once the audio samples were produced, mel-spectrograms for each of the tracks were produced and sorted in relevant categories. Data gathering took longer than anticipated and the project plan had to be modified accordingly.

The experiments revealed a number of challenges. Indeed, the files that aim to be classified are mel-spectrograms, or graphs. Data augmentation revealed to be not suitable to such problem. While data augmentation works well when training a model to identify real life objects, cropping, zooming, shifting and rotating graphs brings little to the learning process as, ultimately, the data that will be passed to the classifier by a user will always be a mel-spectrogram ordered on a x and y axis.

It was attempted to find suitable pre-trained models, but unfortunately research did not yield great results. All pre-trained models available seem to be trained on real-life objects rather than graphs.

These setbacks lead to training one or more models from scratch using only the available data. This, however, allowed for experimenting with multi-class models and binary classifiers. It is indeed an important finding that in the case of small datasets, multiple binary classifiers perform better than one multi-class model.

The usability of the trained models, however, is limited. The main problem is the amount of data required to train the models. While the results obtained are satisfying on the research level, the product could not be released to consumer market without more work. The main issue is that the current model only works

with one specific amplifier. Different amplifiers may have more or fewer settings, and a different amplifier with exactly the same settings may produce a different sound and therefore different mel-spectrograms than the ones the models were trained with.

Another issue that arose is that the amplifier settings can have a very large number of combinations. Currently, the models classify the samples in twelve categories, but to be able to produce an output for every setting individually would require much more data, possibly too much. For instance, the algorithm can only distinguish the equalizer settings from four categories. Since each of the three knobs can have at least ten values, there can be at least a thousand combinations. If all the other settings are considered, it's a grand total of twenty-one thousand combinations possible on the chosen amplifier (provided each knob only ranges from 0 to 10, while in fact values like 1.67 are possible). This may mean that the CNN approach may not be relevant for all users. There is, however, a potential use that is covered in the future works section.

The GUI application is a modern-looking application. It appears that it is suitable for use at the moment, provided a user is interested in classifying audio tracks that were processed with the test amplifier. It fulfils its role of providing an enjoyable user experience, although more work could make it even better.

The main issue with the GUI is the Python implementation. Since the Python scripts require Python 3.8, the user must have the relevant version installed to be able to use the software. A virtual environment is used and included with the software files, but the size of it (around 1 Gb) make it a heavy software for no apparent reason.

5.2 Future Work

Future work may investigate training the models with more data and more categories. This would aim to produce a more relevant output that considers all settings on a chosen amplifier. It would be an interesting experiment to train models with a different amplifier in order to find out whether such models perform well with different devices.

A big part of the future work involves the GUI. Indeed, the current GUI only allows a user to test the models that were trained with the Kuassa Amplikation Lite virtual amplifier. In order to process audio files with such amplifier, a Digital Audio Workstation (DAW) that supports Virtual Studio Technology (VST) is required. It turns out that the official C++ library that allows programmers to develop VST plugins has an open-source C# implementation called VST.NET. This allows for programmers to write their own VST plugins using .NET, but also to write host programs, which involve loading VST plugins into another application. Using such library is rather challenging, especially because the audio must be converted into a specific format and that its implementation requires the use of pointers.

To make the product usable to a broader range of users, it may be useful to investigate the implementation of VST.NET. The GUI could then provide the functionality of scanning a folder containing a number of DI guitar tracks, applying an amplifier effect to them and saving the corresponding mel-spectrogram on disk. Once the process is done, the GUI could then run a ready script to train a model and therefore allow the user to classify future files recorded with this new amplifier. This would make the product useful to individuals who have a home-studio and a great variety of samples available.

Future work may involve implementing the prediction scripts in Python 2.7 in order to run them within IronPython, and therefore removing the need for a user to have all the required libraries and the correct python version pre-installed.

5.3 Research questions, aims and objectives

The research questions were answered.

It is indeed possible to use deep learning to recognize and classify different guitar sounds. It has been shown that it is possible to identify different types of guitar distortions by the means of a convolutional neural network. A graphical user interface was produced to allow a user to easily classify their own audio files.

The project met its aims and objectives.

The available literature on sound recognition and classification was reviewed, as well as literature on deep learning and neural networks. It provided a detailed look into the field of audio processing by the means of artificial intelligence. The literature review provides strong foundations on sound recognition, feature extraction and the types of neural networks that may be used to solve such problems.

A dataset that can be used to train a convolutional neural network was produced. It consists of a number of mel-spectrograms that fall in twelve different categories. The dataset was used successfully to train multiple models.

A convolutional neural network was implemented and produced good to excellent results depending on the model used.

A graphical user interface was developed and can effectively load an audio file, produce a mel-spectrogram for it and run a trained model to classify the mel-spectrogram. The software is modern looking and easy to use.

The quality of the produced output was evaluated by the means of test files. These files were not part of the training dataset nor the validation set and therefore provided a good overview on how well the algorithm performs.

References

Chadabe, J. (1997). *Electric sound.* Prentice Hall.

Cowling, M., & Sitte, R. (2003, November). Comparison of techniques for environmental sound recognition. *Pattern Recognition Letters*, *24*(15), 2895–2907. Retrieved from https://doi.org/10.1016/s0167-8655(03)00147-8 doi: 10.1016/s0167-8655(03)00147-8

Gillespie, D. J., & Ellis, D. P. W. (2013, October). Modeling nonlinear circuits with linearized dynamical models via kernel regression. Retrieved from https://doi.org/10.1109/waspaa.2013.6701830 doi: 10.1109/waspaa.2013.6701830

Goodfellow, I., Bengio, Y., & Courville, A. (2017). *Deep learning.* MIT Press.

Hu, J., Wang, X., Zhang, Y., Zhang, D., Zhang, M., & Xue, J. (2020, jul). Time series prediction method based on variant LSTM recurrent neural network. *Neural Processing Letters*, *52*(2), 1485–1500. Retrieved from https://doi.org/10.1007/s11063-020-10319-3 doi: 10.1007/s11063-020-10319-3

Kim, H.-G., Moreau, N., & Sikora, T. (2004, May). Audio classification based on MPEG-7 spectral basis representations. *IEEE Transactions on Circuits and Systems for Video Technology*, *14*(5), 716–725. Retrieved from https://doi.org/10.1109/tcsvt.2004.826766 doi: 10.1109/tcsvt.2004.826766

LeCun, Y., Bengio, Y., & Hinton, G. (2015, May). Deep learning. *Nature*, *521*(7553), 436–444. Retrieved from https://doi.org/10.1038/nature14539 doi: 10.1038/nature14539

Livieris, I. E., & Pintelas, P. (2019, September). An adaptive nonmonotone active set – weight constrained – neural network training algorithm. *Neurocomputing*, *360*, 294–303. Retrieved from https://doi.org/10.1016/j.neucom.2019.06.033 doi: 10.1016/j.neucom.2019.06.033

MacDonald, G., Godbout, A., Gillcash, B., & Cairns, S. (2019). *Volume-preserving neural networks: A solution to the vanishing gradient problem.*

Retrieved from https://arxiv.org/abs/1911.09576

Naz, S., Umar, A. I., Ahmad, R., Ahmed, S. B., Shirazi, S. H., & Razzak, M. I. (2015, September). Urdu nasta'liq text recognition system based on multi-dimensional recurrent neural network and statistical features. *Neural Computing and Applications*, *28*(2), 219–231. Retrieved from https://doi.org/10.1007/s00521-015-2051-4 doi: 10.1007/s00521-015-2051-4

Rubin, D. (2007). *Inside the blues : 1942 to 1982*. Hal Leonard.

Russell, S. J., & Norvig, P. (2018). *Artificial intelligence* (3rd ed.). Pearson India Education Services Pvt. Ltd.

Schmidhuber, J. (2015, January). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117. Retrieved from https://doi.org/10.1016/j.neunet.2014.09.003 doi: 10.1016/j.neunet.2014.09.003

Schmitz, T., & Embrechts, J.-J. (2018). *Real time emulation of parametric guitar tube amplifier with long short term memory neural network.* Retrieved from https://arxiv.org/abs/1804.07145

Tian, C., Ma, J., Zhang, C., & Zhan, P. (2018, December). A deep neural network model for short-term load forecast based on long short-term memory network and convolutional neural network. *Energies*, *11*(12), 3493. Retrieved from https://doi.org/10.3390/en11123493 doi: 10.3390/en11123493

Wright, A., Damskägg, E.-P., Juvela, L., & Välimäki, V. (2020, January). Real-time guitar amplifier emulation with deep learning. *Applied Sciences*, *10*(3), 766. Retrieved from https://doi.org/10.3390/app10030766 doi: 10.3390/app10030766

Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine*, *13*(3), 55-75. Retrieved from https://doi.org/10.1109/mci.2018.2840738 doi: 10.1109/mci.2018.2840738

Zhang, Z., Olbrych, E., Bruchalski, J., McCormick, T. J., & Livingston, D. L. (2018). *A vacuum-tube guitar amplifier model using long/short-term memory networks.* Retrieved from https://doi.org/10.1109/SECON.2018.8479039 doi: 10.1109/SECON.2018.8479039

Appendices

A Initial Project Overview

Initial Project Overview

SOC10101 Honours Project

1. Title of Project

Neural network to analyse and classify electric guitar sounds depending on the type of distortion applied to them.

2. Problem and Objectives

The overall problem relates to audio recognition and classification by a neural network.

The classification is likely to happen using visual representations of the audio spectrum and a convolutional neural network.

The objectives are to first identify which filters will allow the CNN to extract features relevant to the classification wanted.

Once figured out, the algorithm will need to be trained and tested for accuracy and precision.

The algorithm is required to return the settings needed to achieve a similar guitar sound to the user.

Additionally, and if time permits, a GUI could be implemented as so to make the whole process user friendly.

3. Investigation

The problem context will be investigated using tools such as Google Scholar, Napier University's library, IEEE Xplore, ProQuest and other academic paper databases. The secondary sources available for this are journal articles, reviews, books or academic publications. An emphasis will be put on material that is peer reviewed to ensure it is of the right standard for an honours project.

4. Empirical data collection

Empirical data will be collected through experiment with neural networks.

It can be as simple as collecting the output given by the algorithm but will also consist of test results obtained following a rigorous testing plan in order to analyse the efficacy of the algorithm.

5. Deliverable

The intention is to produce an algorithm that will recognize the type of guitar distortion when given a sample audio track and return to the user the amplifier settings needed to achieve a similar sound.

Such product would benefit musicians and/or music producers looking to recreate a specific guitar sound. It could be used for recreational purposes as well as in professional environments (e.g. recording studios) in order to reduce the time and effort needed to obtain the wanted sound.

6. Development Lifecycle

Careful planning will be undertaken in order to define realistic and achievable requirements.

An analysis of current implementations of neural networks is required to establish how such algorithms can or cannot help producing the wanted result. It will identify the pros and cons of such algorithms, highlight potential issues and provide more insight on how such system is being managed.

The implementation will likely take the form of a python scripts to which an audio track is passed to and produce an output. If time allows it, a simple GUI could be implemented to make the product easy to use.

7. Technical/Analytical skills

In order to ensure a positive outcome for that project critical analysis skills will need to be developed with the aim of identifying aspects of neural network development that are relevant to the project.

Technical skills in implementing such system in python will need to be developed to ensure the project works.

8. Key Challenges

The chosen way of implementing such system is using a convolutional neural network to analyse visual representations of sounds. A possible problem could appear if no efficient filters are found to allow for classification. This problem will be managed by changing and adapting the wanted outcome to a more generic output, such as classifying sounds from ‘clean guitar’ to ‘heavy distortion’.

B Second Formal Review Output

SOC10101 Honours Project (40 Credits)

Week 9 Report

Student Name: Maxime Greffe

Supervisor: Dimitra Gkatzia

Second Marker: Simon Powers

Date of Meeting: 30/11/2020

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? **yes**

If not, please comment on any reasons presented

Please comment on the progress made so far

Very good literature review showing excellent understanding of the techniques to be used. Good to see such clear explanations and good understanding of deep learning. It would be good to build on this by further developing the practical work as soon as possible.

Is the progress satisfactory? **yes**

Can the student articulate their aims and objectives? **yes**

If yes then please comment on them, otherwise write down your suggestions.

Aims and objectives are very suitable for an honours project – no concerns here.

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

Does the student have a plan of work? **yes**

If yes then please comment on that plan otherwise write down your suggestions.

Plan was feasible.

Does the student know how they are going to evaluate their work? **yes**

If yes then please comment otherwise write down your suggestions.

It was agreed that the evaluation could focus on the technical aspects of the work, i.e. evaluating the performance of the machine learning algorithms (although the application should still be developed to meet the programme requirements).

Any other recommendations as to the future direction of the project

The research questions could be refined to make them non binary, i.e. to what extent...

A thorough statistical analysis of the results needs to be performed as the training algorithms are stochastic. Use of appropriate stats (e.g. statistical significance tests) will be needed to get a first class mark (which you should aim for!). Enough computational time should be allowed to run multiple replicates to allow for this.

Overall a really interesting project!

Signatures: Supervisor Second Marker Simon Powers

Student

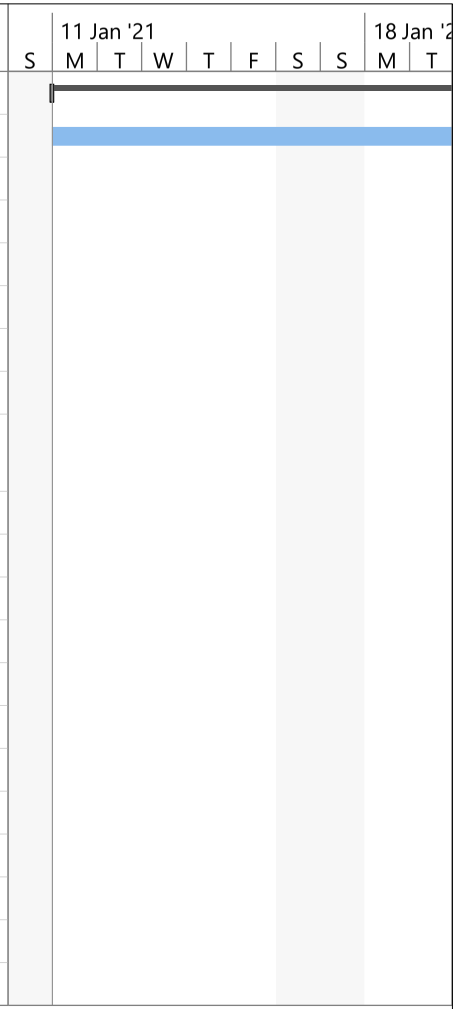
* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

The student should submit a copy of this form to Moodle immediately after the review meeting; A copy should also appear as an appendix in the final dissertation.

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

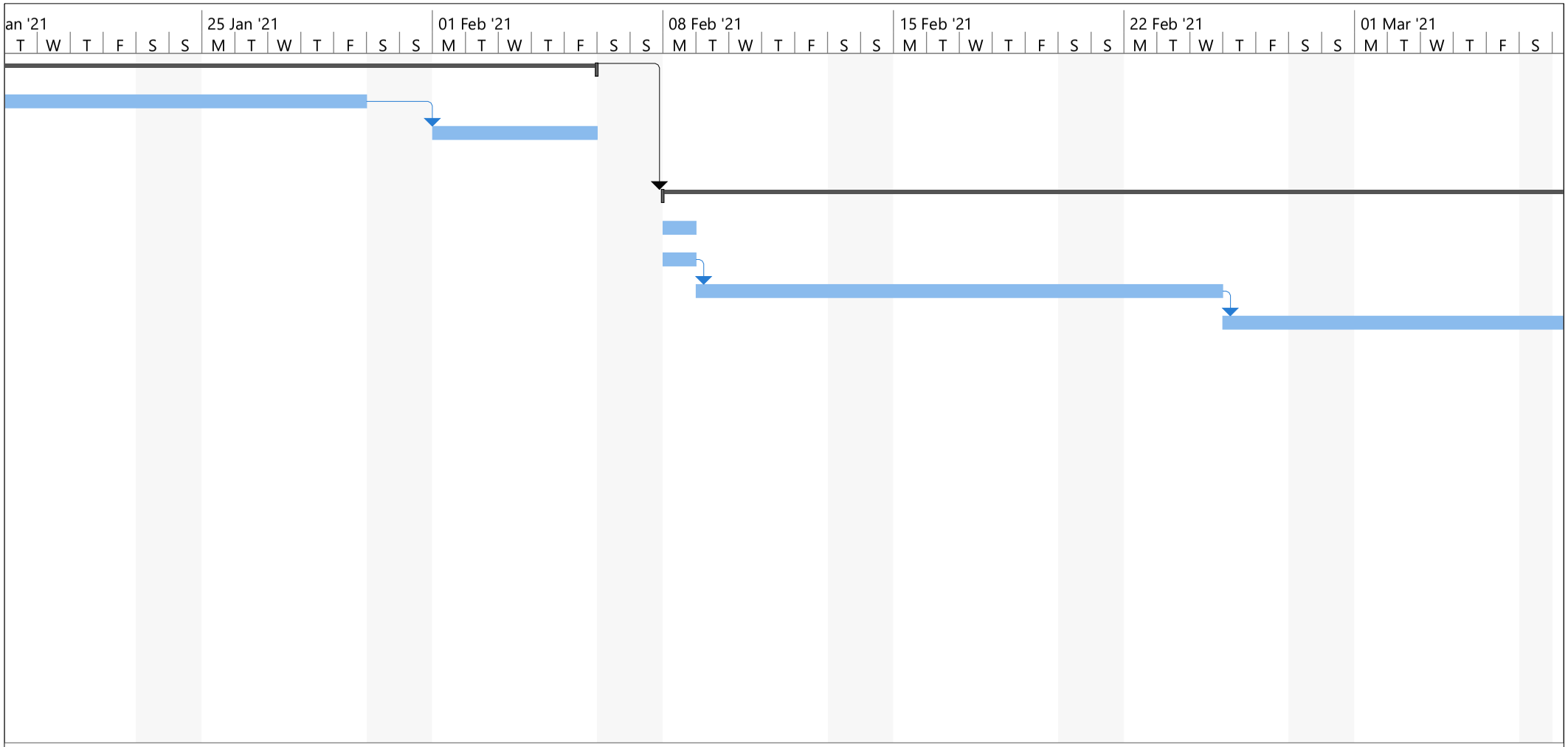
C Project Plan

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1		**Prepare training data**	**20 days**	**Mon 11/01/21**	**Fri 05/02/21**		
2		Gather audio files	15 days	Mon 11/01/21	Fri 29/01/21		
3		Produce spectral graphs	5 days	Mon 01/02/21	Fri 05/02/21	2	
4							
5		**CNN Development**	**23 days**	**Mon 08/02/21**	**Wed 10/03/21**	**1**	
6		Review requirements	1 day	Mon 08/02/21	Mon 08/02/21		
7		Choose library	1 day	Mon 08/02/21	Mon 08/02/21		
8		Implement CNN	12 days	Tue 09/02/21	Wed 24/02/21	7	
9		CNN testing and evaluation	10 days	Thu 25/02/21	Wed 10/03/21	8	
10							
11		**GUI Development**	**17 days**	**Thu 11/03/21**	**Fri 02/04/21**	**5**	
12		Review requirements	1 day	Thu 11/03/21	Thu 11/03/21		
13		Implement GUI	16 days	Fri 12/03/21	Fri 02/04/21	12	
14							
15		**Testing**	**6 days**	**Mon 05/04/21**	**Mon 12/04/21**	**11**	
16		Unit testing	3 days	Mon 05/04/21	Wed 07/04/21		
17		Black Box Testing	3 days	Thu 08/04/21	Mon 12/04/21	16	
18							
19		**Write Documentation**	**13 days**	**Mon 05/04/21**	**Wed 21/04/21**	**11**	
20		Results	7 days	Mon 05/04/21	Tue 13/04/21		
21		Evaluation	6 days	Wed 14/04/21	Wed 21/04/21	20	



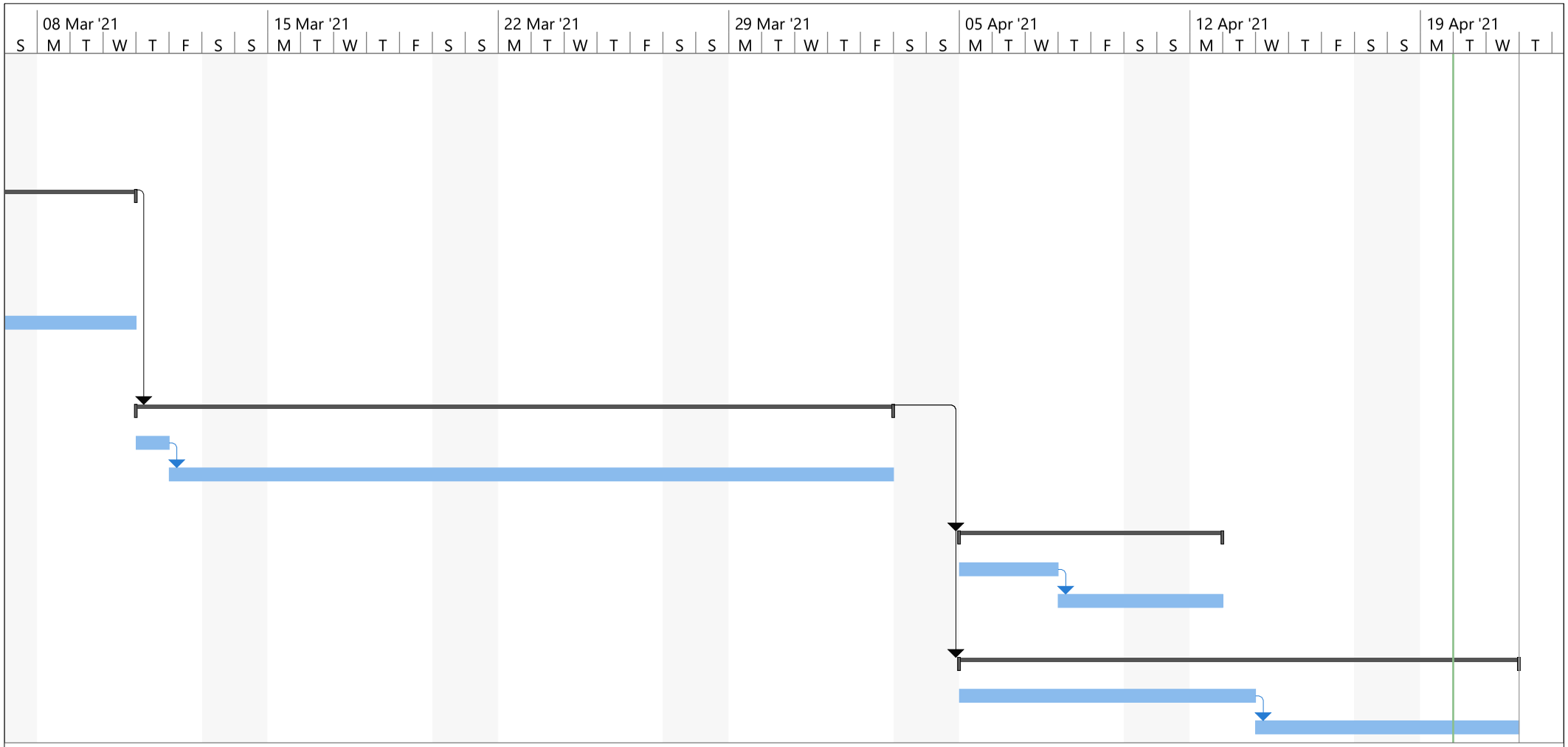
Project: Project Plan
Date: Tue 20/04/21

Task
Split
Milestone
Summary
Project Summary
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
External Tasks
External Milestone
Deadline
Progress
Manual Progress



Project: Project Plan
Date: Tue 20/04/21

Task	Inactive Summary	External Tasks
Split	Manual Task	External Milestone
Milestone	Duration-only	Deadline
Summary	Manual Summary Rollup	Progress
Project Summary	Manual Summary	Manual Progress
Inactive Task	Start-only	
Inactive Milestone	Finish-only	



Project: Project Plan
Date: Tue 20/04/21

Task		Inactive Summary		External Tasks	
Split		Manual Task		External Milestone	
Milestone		Duration-only		Deadline	
Summary		Manual Summary Rollup		Progress	
Project Summary		Manual Summary		Manual Progress	
Inactive Task		Start-only			
Inactive Milestone		Finish-only			

D Diary Sheets

EDINBURGH NAPIER UNIVERSITY
SCHOOL OF COMPUTING
PROJECT DIARY

Student: Maxime Greffe **Supervisor:** Dimitra Gkatzia

Date: 29/09/20 **Last diary date:** -

Objectives:

To bring to the next meeting:
Layout of literature review: sections and subsections
Find papers to base my thesis on

BEng Software Engineering requirements:
technology review - discuss libraries that I will use
machine learning libraries
libraries to turn sound into images

Progress:

Supervisor's Comments:

EDINBURGH NAPIER UNIVERSITY
SCHOOL OF COMPUTING
PROJECT DIARY

Student: Maxime Greffe **Supervisor:** Dimitra Gkatzia

Date: 13/10/20 **Last diary date:** 29/09/20

Objectives:

To bring to the next meeting:
Swap section 2 and section in literature review
Write up section 3 (which is now section 2) so Dimitra can provide advice on how to solve the problem (send draft a couple of days before the meeting)
Check auto-encoders

Progress:

Literature review going well, according to plan

Supervisor's Comments:

Good so far

Need a Gantt chart by week 9

EDINBURGH NAPIER UNIVERSITY
SCHOOL OF COMPUTING
PROJECT DIARY

Student: Maxime Greffe **Supervisor:** Dimitra Gkatzia

Date: 27/10/20 **Last diary date:** 13/10/20

Objectives:

Keep going with literature review.
Add references where feedback was provided.

Progress:

Literature review going well, according to plan

Supervisor's Comments:

Add references where feedback was provided.

EDINBURGH NAPIER UNIVERSITY
SCHOOL OF COMPUTING
PROJECT DIARY

Student: Maxime Greffe **Supervisor:** Dimitra Gkatzia

Date: 10/11/20 **Last diary date:** 27/10/20

Objectives:

To add to literature review

intro:
aims and objective of the research
research questions

overview of the whole project
motivate my work, tell why I am doing that

describe project as much as possible, mention what data I'm going to use

(couple of pages)

make figures clearer, add captions

have another section about previous work that has looked into classifying guitar sounds

produce a Gantt chart

Progress:

Literature review going well, according to plan

Supervisor's Comments:

On good track

EDINBURGH NAPIER UNIVERSITY
SCHOOL OF COMPUTING
PROJECT DIARY

Student: Maxime Greffe **Supervisor:** Dimitra Gkatzia

Date: 24/11/20 **Last diary date:** 10/11/20

Objectives:

Literature Review:

swap problem statement and intro bit
reduce the scope of the project (reproduction optional)
remove professional part from research questions
research tutorials to recreate sounds and see if it is realistic to reproduce sounds
write up a conclusion, that motivates my work

Project Plan:
require more time for the evaluation (can run in parallel with other tasks)
If musicians will be testing it, you need to account for scheduled meetings

Interim Report Meeting:
next week meeting with 2nd marker
email second marker

Progress:

- swap problem statement and intro bit DONE
- reduce the scope of the project (reproduction optional) DONE
- remove professional part from research questions DONE
- research tutorials to recreate sounds and see if it is realistic to reproduce sounds DONE (not realistic, removed this requirement)
- write up a conclusion, that motivates my work

Project Plan: Changes done

email second marker

Supervisor's Comments:

good

EDINBURGH NAPIER UNIVERSITY
SCHOOL OF COMPUTING
PROJECT DIARY

Student: Maxime Greffe **Supervisor:** Dimitra Gkatzia

Date: 22/03/21 **Last diary date:**

Objectives:

remove 3rd research question -

add aims and objectives between 2 and 3:
create a dataset that can be used to train a CNN

conclusion lit rev:
why do I use a CNN?, why do I use visual representations?
motivate my choices

methodoloy and implementation

between methodology and pre-development,
give an overview of the whole project
give an overview of this section

diagram of the methodology
(steps taken to reach outcome, e.g. gather data, implement CNN)

3.2 dataset: describe what the amplifier channels are (and other settings)

Progress:

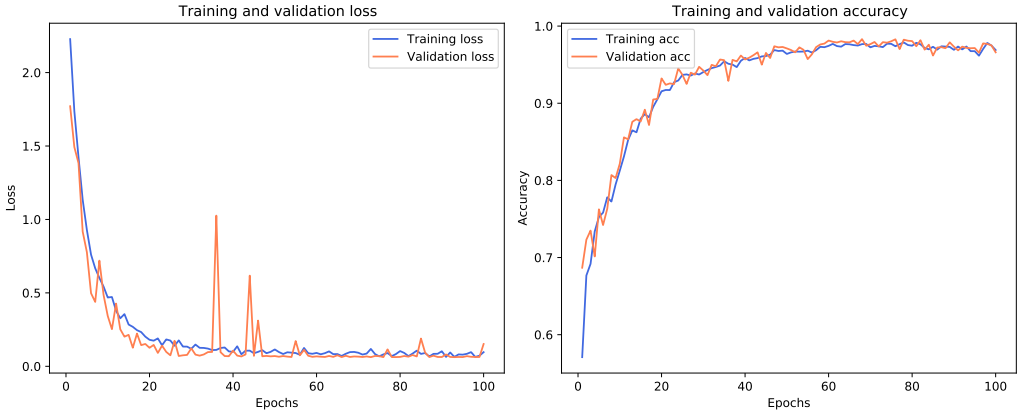
Good progress.

Supervisor's Comments:

Focus on my report and email Dimitra

E Performance Graphs and Test Outputs of All Models

E.1 Multi-class model

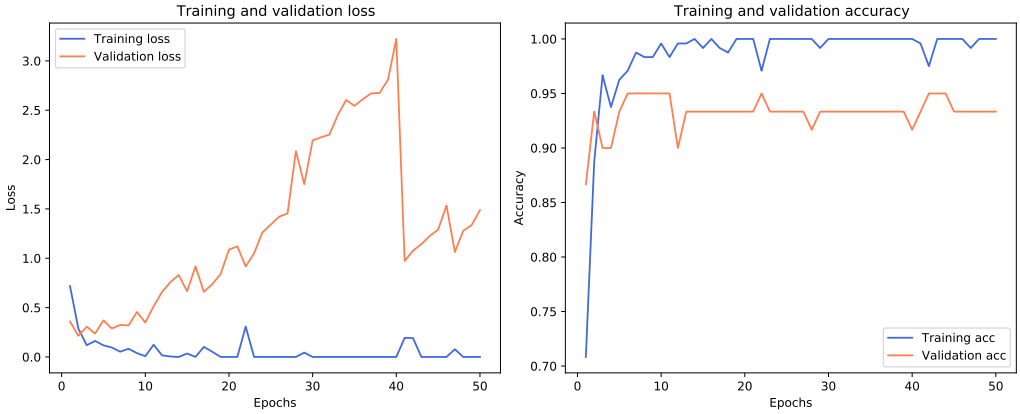


Pass: 81.67% (98), Fail: 18.33% (22), Total: 120

Class	Precision	Recall	F-Score
clean-0-0-10	62.0	100.0	0.77
clean-0-10-0	100.0	100.0	1.0
clean-10-0-0	77.0	100.0	0.87
clean-5-5-5	100.0	70.0	0.82
lead-0-0-10	100.0	40.0	0.57
lead-0-10-0	77.0	100.0	0.87
lead-10-0-0	100.0	70.0	0.82
lead-5-5-5	100.0	100.0	1.0
rhythm-0-0-10	100.0	100.0	1.0
rhythm-0-10-0	50.0	100.0	0.67
rhythm-10-0-0	0.0	0.0	0
rhythm-5-5-5	100.0	100.0	1.0

E.2 Binary Classifier

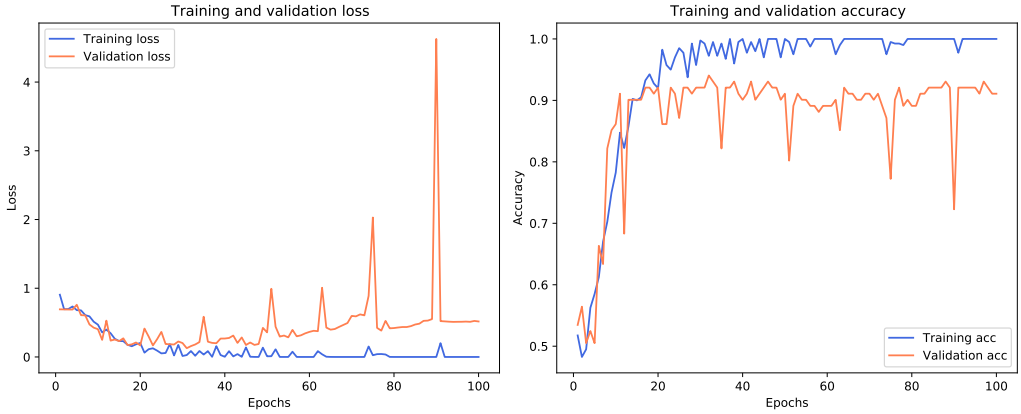
Amp Channel - ”Clean” VS ”Lead or Rhythm”




```
Pass: 100.0% (60), Fail: 0.0% (0), Total: 60
+-----------------+-----------+--------+---------+
|      Class      | Precision | Recall | F-Score |
+-----------------+-----------+--------+---------+
|      clean      |   100.0   | 100.0  |   1.0   |
| lead or rhythm  |   100.0   | 100.0  |   1.0   |
+-----------------+-----------+--------+---------+
```

E.3 Binary Classifier

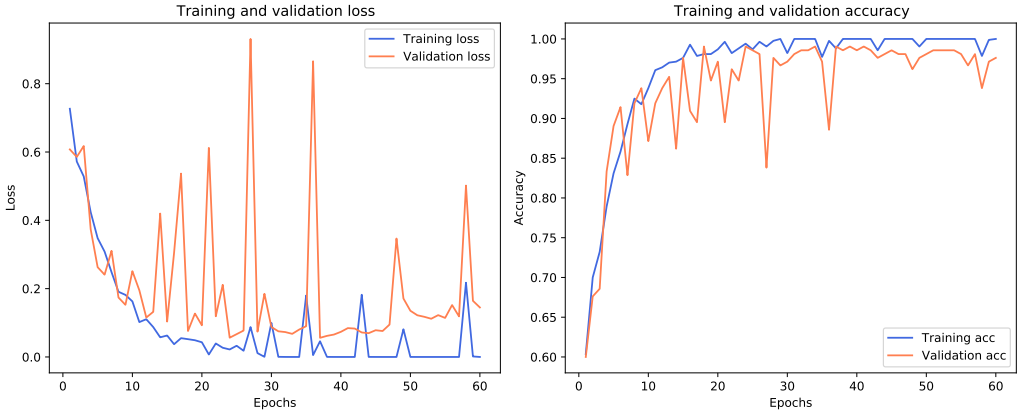
Amp Channel - ”Lead” VS ”Rhythm”




```
Pass: 92.0% (92), Fail: 8.0% (8), Total: 100
+--------+-----------+--------+---------+
| Class  | Precision | Recall | F-Score |
+--------+-----------+--------+---------+
|  lead  |    94.0   |  90.0  |   0.92  |
| rhythm |    90.0   |  94.0  |   0.92  |
+--------+-----------+--------+---------+
```

E.4 Binary Classifier

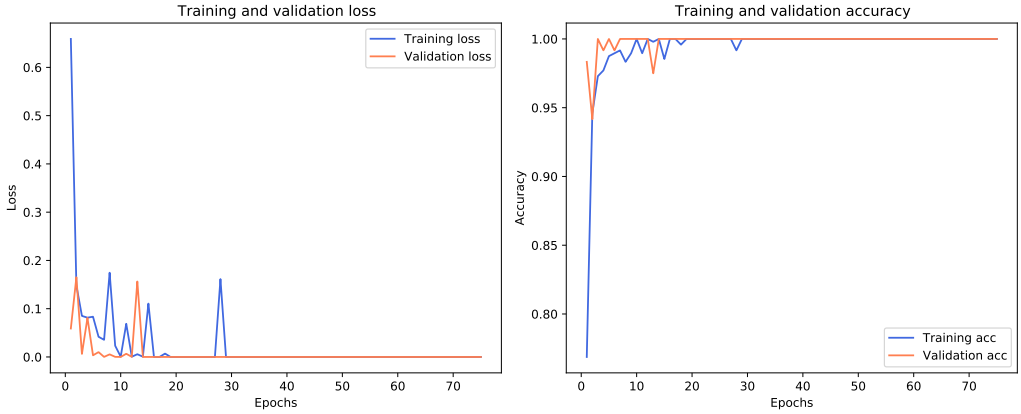
Equalizer Settings - ”10-0-0 or 0-0-10” VS ”0-10-0 or 5-5-5”




```
Pass: 98.56% (206), Fail: 1.44% (3), Total: 209
+------------------+-----------+--------+---------+
|      Class       | Precision | Recall | F-Score |
+------------------+-----------+--------+---------+
| 0-10-0 or 5-5-5  |    99.0   |  98.0  |   0.98  |
| 10-0-0 or 0-0-10 |    98.0   |  99.0  |   0.98  |
+------------------+-----------+--------+---------+
```

E.5 Binary Classifier

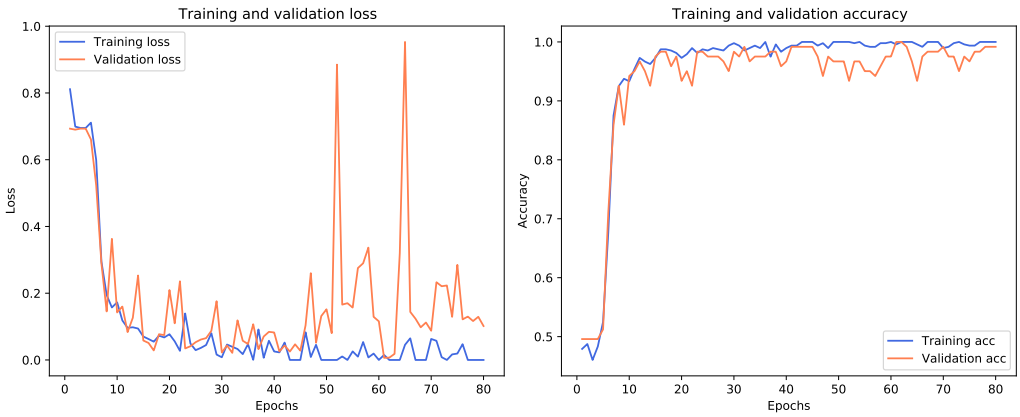
Equalizer Settings - ”10-0-0” VS ”0-0-10”




```
Pass: 100.0% (120), Fail: 0.0% (0), Total: 120
+--------+-----------+--------+---------+
| Class  | Precision | Recall | F-Score |
+--------+-----------+--------+---------+
| 0-0-10 |   100.0   | 100.0  |   1.0   |
| 10-0-0 |   100.0   | 100.0  |   1.0   |
+--------+-----------+--------+---------+
```

E.6 Binary Classifier

Equalizer Settings - ”0-10-0” VS ”5-5-5”




```
Pass: 98.33% (118), Fail: 1.67% (2), Total: 120
+--------+-----------+--------+---------+
| Class  | Precision | Recall | F-Score |
+--------+-----------+--------+---------+
| 0-10-0 |    97.0   | 100.0  |   0.98  |
| 5-5-5  |   100.0   |  97.0  |   0.98  |
+--------+-----------+--------+---------+
```

F Python Virtual Environment Requirements.txt

```
absl-py==0.12.0
appdirs==1.4.4
astunparse==1.6.3
audioread==2.1.9
backcall==0.2.0
cachetools==4.2.1
certifi==2020.12.5
cffi==1.14.5
chardet==4.0.0
colorama==0.4.4
cycler==0.10.0
decorator==5.0.6
flatbuffers==1.12
gast==0.3.3
google-auth==1.28.0
google-auth-oauthlib==0.4.3
google-pasta==0.2.0
grpcio==1.32.0
h5py==2.10.0
idna==2.10
ipython==7.22.0
ipython-genutils==0.2.0
jedi==0.18.0
joblib==1.0.1
Keras==2.4.3
Keras-Preprocessing==1.1.2
kiwisolver==1.3.1
librosa==0.8.0
llvmlite==0.36.0
Markdown==3.3.4
matplotlib==3.4.1
numba==0.53.1
numpy==1.19.5
oauthlib==3.1.0
opt-einsum==3.3.0
packaging==20.9
parso==0.8.2
pickleshare==0.7.5
Pillow==8.1.2
pooch==1.3.0
prompt-toolkit==3.0.18
protobuf==3.15.6
pyasn1==0.4.8
pyasn1-modules==0.2.8
pycparser==2.20
Pygments==2.8.1
pyparsing==2.4.7
python-dateutil==2.8.1
PyYAML==5.4.1
requests==2.25.1
requests-oauthlib==1.3.0
resampy==0.2.2
rsa==4.7.2
scikit-learn==0.24.1
scipy==1.6.1
six==1.15.0
SoundFile==0.10.3.post1
tensorboard==2.4.1
tensorboard-plugin-wit==1.8.0
tensorflow==2.4.1
tensorflow-estimator==2.4.0
termcolor==1.1.0
threadpoolctl==2.1.0
traitlets==5.0.5
typing-extensions==3.7.4.3
urllib3==1.26.4
wcwidth==0.2.5
Werkzeug==1.0.1
wrapt==1.12.1
```