TEMPO ESTIMATION OF ELECTRONIC MUSIC USING MACHINE LEARNING ALGORITHMS TRAINED ON A DJ'S OWN MUSIC LIBRARY

William Caine – Student No. 160263260

11/05/2020

I190 – BSc Hons. Computer Science (Security and Resilience)

Supervised by Dr. Jaume Bacardit

Word count: 7,951

ABSTRACT

Tempo estimation of electronic music is a common problem in the world of digital DJing. Many algorithms exist to accomplish this task and most DJ software tools make use of such algorithms. However, these algorithms are not always as accurate as desired. As such, a DJ often must manually check and annotate tempo values for each track in their library. As tempo estimation is a problem of pattern recognition, and most DJs have an existing library of annotated tracks; I propose that training a machine learning algorithm using this library may offer an alternative solution. While prior research seems to suggest that this may be viable, I was ultimately unable to replicate their success in training a model that was accurate enough to be practically useful.

DECLARATION

I declare that this dissertation represents my own work except where otherwise stated.

ACKNOWLEDGEMENTS

I would like to thank the ever helpful and patient Dr. Jaume Bacardit who supervised this project. Without his guidance and suggestions this project would not have been possible.

I would also like to thank my colleagues at Scott Logic, and in particular Chris Price, who helped to inspire this project. During my internship with them in the summer of 2018 I gained valuable first-hand experience working professionally as a software developer, as well as working with machine learning for the first time.

My close friends, Joshua Herbert (Decizion) and Joseph Nicholls (Aperio), were also very helpful throughout, providing feedback and support from the perspective of professional DJs.

Lastly, I would like to thank my very patient father, Robert, for playing the role of "rubber duck" throughout this project as well as my brother's fiancée, Sarah, for her proof-reading and technical writing guidance.

TABLE OF CONTENTS

1. INTRODUCTION

As a part-time DJ using modern digital tools, I often found myself frustrated with the amount of time that needed to be spent preparing my music on the computer before a performance. One of the most time-consuming parts of this preparation is annotating each track with its tempo, measured in beats per minute (BPM), to aid in live mixing during a performance. Most DJ software packages come with tools to automatically estimate this value, however the accuracy of these tools is not always ideal. This leaves the DJ with inaccurate annotations or requires them to manually check the results. There are a variety of approaches and algorithms used to accomplish tempo estimation, but not many of the publicly available methods appear to use machine learning. Although there are several commercial software packages that keep their methodology a closely guarded secret.

With this in mind, I set out to assess the potential of using machine learning to accomplish this task and to see if it could be a viable alternative to the existing algorithms. The task involves identifying patterns in the timing of the music, particularly by identifying percussive elements, and so this pattern recognition struck me as something that a machine learning algorithm may be suited to. Additionally, as a DJ myself, I already had access to a fairly large dataset of partially annotated music that could be used as training data, as do many other DJs.

1.1 Document Outline

Chapter 1 – Introduction
An introduction to this project, covering the motivation and rationale for the research, as well as the specific aims and objectives to be completed.

Chapter 2 - Background
Background information on existing solutions, machine learning, neural networks, and the technologies used throughout this project. Additionally, I review some previous research on the subject.

Chapter 3 - Developing a Solution
A detailed account of the process I went through in order to develop and produce: the training data, neural network architecture, and experimental models for this project.

Chapter 4 - Results and Evaluation
The results of my experiments and evaluation of how well the models performed; by comparison to each other, previous research, and alternative solutions.

Chapter 5 - Conclusion
A final review of the project, as per the initial aims and objectives, as well as my proposed future work.

1.2 Aims and Objectives

In my original project proposal, I stated that I wished to predict not only the BPM of the music, but also the onset of the first beat, as these are both properties of its tempo. Using these two values in conjunction, you can determine the exact time at which each beat in the track occurs. As the project progressed however, I found predicting these two values was adding a lot of complexity. As such, I decided to shift my focus to just predicting the BPM alone, as this is the most useful value to know and I believe the success of this model would be a good indication of whether further research would be worthwhile. My hypothesis for this project was therefore:

Existing tempo estimation algorithms could be improved using machine learning algorithms trained on a DJ's own music library.

In order to test my hypothesis, I laid out the following objectives:

- Collect and label a large enough dataset of music samples
- Investigate methods of pre-processing the audio
- Research different types of neural network and their uses
- Train and evaluate my own neural network models
- Compare the accuracy of my models to existing algorithms

2. BACKGROUND

In order to undertake this project, I would need to educate myself on the topic of machine learning and specifically neural networks. I also researched existing methods of tempo estimation and previous works on the topic.

2.1 Machine Learning

Machine learning (ML) is the term used to describe the process of building mathematical models, without explicit instructions, to infer patterns and features in data [1]. It is considered a branch of artificial intelligence (AI) and is an increasingly hot topic for research in recent years. Machine learning is quite a broad term to describe a variety of approaches to building such models, but it is becoming a very commonly used technique for tasks such as computer vision and natural language processing, where traditional algorithms can struggle to achieve the desired results.

2.1.1 Supervised Learning

Supervised learning is a specific style of machine learning that makes use of an existing dataset of annotated examples [2]. This dataset is referred to as "training data" and consists of a set of examples, each annotated with its desired output.

Using an iterative approach to optimisation of an objective function, the model is tuned to produce a new function that is able to predict the output of previously unseen inputs. An optimal algorithm will successfully be able to predict the output of examples that were not a part of the training data. The process of improving the accuracy of this algorithm is what we refer to as machine learning [3].

Supervised learning algorithms include classification and regression types. Classification algorithms are used to produce an output belonging to a discrete set of possible values. Regression algorithms, on the other hand, are used to produce a single continuous value.

2.1.2 Artificial Neural Networks

Artificial neural networks (ANNs) are one approach to implementing a supervised learning algorithm. The structure of these networks is inspired by biological neural networks in brains [4]. The network consists of a series of connected nodes called artificial neurons. These neurons are loosely based on biological cells, taking in an input signal, processing it, and potentially signalling other connected neurons.

This "signal" is typically represented by a real number and the processing done by each neuron to this signal is usually some non-linear function. Each neuron typically has a weight to it that increases or decreases the signal strength. This weight is adjusted during learning to tune the algorithm.

Neurons tend to be aggregated into layers to perform various transformations or functions. An input is passed to the input layer of the network, processed through some number of hidden layers, and finally an output is given by the output layer.

A classic example of this type of network might be used for image recognition. For example, a network could be built to identify the presence of a cat in an image. The input in this case would be an image, represented by a multi-dimensional array of pixel values. The output might consist of 2 neurons, one representing "cat" and the other representing "no cat". Whichever neuron has the higher output signal would be considered the prediction.

2.1.3 Convolutional Neural Networks

A convolutional neural network (CNN) is a type of ANN that takes direct inspiration [5] from observations made about how biological brains process visual stimuli [6]. A typical CNN is constructed using "convolutional" layers. These work by convolving over the image with a series of filters to identify features. CNNs are very commonly used in machine learning for interpreting visual imagery, such as image classification or medical imagery analysis.

Each convolutional layer in a CNN has a couple of important hyper-parameters to consider. First is the number of filters to convolve over the image with. Second is the kernel size, this defines the shape of each filter in width × height. As such, a convolutional layer will accept, as input, a tensor of shape (image width) × (image height) × (image depth) and produce a feature map output of shape (number of filters) × (feature map width) × (feature map height) × (image depth). A feature map produced by one of these convolutions can be treated as an image itself and passed as input to another convolutional layer to perform even more abstract feature detection.

Since a convolutional layer increases dimensionality, with each filter producing its own feature map, convolutional layers are typically followed by a pooling layer. A pooling layer works to reduce information by reducing each (pool width) × (pool height) set of values in the feature map down to a single value. This might be done by taking the maximum or average across all the values in the pool to produce the result.

2.1.4 Dropout

When training ANNs, we often come up against a problem known as over-fitting. This is where the network learns to look for very specific traits present in the training data, that do not necessarily correlate with any meaningful information. There are many approaches to reducing over-fitting, primarily by using a broad and varied set of training data and a separate set of data, known as the validation set, to validate performance.

Another technique used to reduce overfitting, however, is the use of dropout layers in network designs. These layers are fairly simple in concept but have been shown to effectively reduce over-fitting [7]. They work by simply temporarily ignoring some neurons, and all their connections, based on a specified probability. This helps to build a more robust model by encouraging the model to share responsibility across its neurons and reducing co-adaption of neurons across layers.

2.1.5 Batch Normalisation

Batch normalisation is a technique used in ANNs to accelerate and improve learning [8]. It is performed by introducing batch normalisation layers into the model that normalise values produced by the previous layers. This is accomplished by subtracting the batch mean and dividing by the batch standard deviation. These values are referred to as gamma and beta, respectively, and become trainable parameters themselves. This is to allow the optimisation algorithm to adjust the level of normalisation and reverse any negative effects of it.

Batch normalisation has been shown to drastically reduce the time needed for training by compensating and allowing for larger learning rates and less carefully initialised values.

2.2 Previous Works

2.2.1 A Single Step Approach to Musical Tempo Estimation Using a CNN

In ISMIR conference proceedings [9], research with a very similar goal to that of this project was conducted. The purpose of this paper was to develop a CNN model that could estimate the tempo of music directly from a mel-spectrogram. This is contrary to existing solutions, such as the ones that will be discussed in section 2.3, which typically try to detect the onset of beats and derive a BPM value from the time intervals between them.

In their research, they use convolutional layers with 1×5 kernels oriented along the time-axis of the spectrograms with the purpose of detecting beat onset. Then, they use a combination of much longer kernels, up to the full length of the spectrogram image, in a series of multi-filter modules. These longer-kernel convolutional layers are preceded by an average pooling layer with the pool oriented along the frequency axis. They propose that doing so will act as a summarisation of the frequency spectrum before detecting temporal features over longer time periods with the convolutional layers.

Interestingly, the model used is a classification approach, rather than regression. As BPM is a continuous numerical value, the intuitive approach might be to use a regression model for this task. However, the research notes that, in informal experiments, a classification approach was determined to be more successful. This is something I wanted to test myself experimentally.

2.2.2 GiantSteps Dataset

A dataset, published at the 16th International Society for Music Information Retrieval Conference [10], contains tempo and key annotated electronic music tracks. This is one of very few such datasets and is used in the above research to represent electronic music.

The *Beatport* website[1] is an online record store targeted at DJs, with a focus on electronic music. Each track listed for sale on the website is annotated with a BPM, but these are not always correct. Therefore, this dataset was formed by looking at submitted user corrections to BPMs of tracks listed on the website, as this was considered to be a good source of ground truth.

The dataset comprises of 664 tracks, of which 1-minute samples can be freely downloaded from the *Beatport* website. This could provide an ideal dataset for training and evaluating tempo estimation models, however the purpose of this project was to use a DJ's own music library as training data.

In the proceedings, it is also noted that it is common to assess tempo estimation algorithms using two metrics. Accuracy1 considers an estimate correct if it falls within ±4% of the true tempo. Accuracy2 additionally considers an estimate correct if it falls within ± 4% of the true tempo multiplied by $\frac{1}{3}$, $\frac{1}{2}$, 2, or 3. This allows some tolerance of so-called "octave errors" where the derived BPM is incorrect by one of these factors. This is something I have noticed a lot anecdotally when analysing my own music. These metrics will be useful when evaluating my own models' performance in chapter 4.

Using these metrics and dataset, a set of benchmarks were published in the proceedings that show how well various algorithms and commercial software performs. I will refer to these benchmarks later in my results and evaluation in chapter 4.

[1] https://beatport.com/

Algorithm	accuracy1	accuracy2
Beatport (2014 website)	4.819%	23.795%
Davies and Plumbley [TASLP 2007]	29.367%	48.042%
Böck et al. [ISMIR 2015] (range: 50-240)	56.325%	88.253%
Böck et al. [ISMIR 2015] (range: 95-190)	76.506%	86.597%
Böck et al. [ISMIR 2015] (range: 88-175)	69.289%	85.693%
Gkiokas et al. [ICASSP 2012]	58.886%	82.380%
Percival and Tzanetakis [TASLP 2014]	51.355%	88.404%
Hörschläger et al. [SMC 2015]	75.000%	82.831%
Schreiber and Müller [ICASSP 2014]	56.928%	87.048%
Deckadance (range: 80+)	57.681%	81.627%
CrossDJ Free (range: 75-150)	63.404%	90.211%
NI Traktor PRO 2 (trial) (range: 60-200)	64.608%	88.705%
NI Traktor PRO 2 (trial) (range: 88-175)	76.958%	88.705%
Pioneer Rekordbox v3.2.2	74.548%	89.157%
beaTunes v4.5.5 (no online, OnsetPeak)	56.175%	85.843%
beaTunes v4.5.5 (no online, OnsetPeak, range: 90-180)	75.753%	85.693%
beaTunes v4.5.5 (no online, OnsetPeak, genre tags embedded)	76.807%	85.693%

Table 1 Amended GiantSteps benchmark results taken from their website[2] on 04/05/2020.

2.3 Existing Solutions

Several algorithms and solutions already exist to solve the problem of tempo estimation, many of which are noted in table 1. However, as we can see, these solutions are not always accurate and there is room for improvement. In general, most of these algorithms work to detect the onset of musical beats, deriving the BPM from the time between these beats. However, a common problem with this approach is the "octave errors" described above. Some of the basic principles of these algorithms are as follows.

2.3.1 Sound Energy Algorithm

The most basic algorithm for detecting BPM uses "sound energy" to identify beats [11]. This works by splitting the audio into blocks, calculating the energy of each block and comparing it to that of a window of preceding blocks. If block *j* were to contain 1,024 samples in stereo, then its energy could be calculated as follows:

$$E_j = \sum_{i=0}^{1023} left[i]^2 + right[i]^2$$

Blocks above a certain threshold of energy are determined to contain a beat.

2.3.2 Low Pass Filter Algorithm

One potential way to improve this sound energy algorithm, would be to first process the audio with a low pass filter [12]. A low pass filter removes the higher frequency elements of the music such as melodic and harmonic parts, leaving the lower frequencies. This works because most music,

[2] http://www.cp.jku.at/datasets/giantsteps/

particularly western music, contains percussive elements in this register that are indicative of a beat. For example, a kick drum. These are quite primitive methods, however.

2.3.3 Fourier Transforms

Another way of pre-processing the audio could be to use a Fourier transform. This is a way of representing the original signal as a sum of basis sine and cosine functions, each representing different frequencies. This reduces the amount of information and enables us to represent it in a simplified way.

2.3.4 Discrete Wavelet Transforms

Alternatively, another method of transformation, more commonly used in tempo estimation algorithms [13], is the discrete wavelet transform (DWT). A DWT is a wavelet transform for which wavelets are discretely sampled, as opposed to using basis functions as in Fourier transforms. This has the benefit of improved temporal resolution, by capturing both frequency and location (time) information. Understandably so, an increased temporal resolution is beneficial in tempo estimation tasks.

2.4 Software Libraries

In order to develop my solution, I needed to decide which software stack and libraries I was going to use. This is not an exhaustive list of every software package used, see appendix A for the full source code.

2.4.1 TensorFlow

Prior to this project I did an internship with a local software consultancy. During my time there, I developed an ML model. For that project, we used the TensorFlow library [14]. As I was already somewhat familiar with this library, I was keen to use it again but thought it was worth searching online to see if any other alternatives were available.

There were a few options available from cloud computing services such as Amazon Web Services and Google Cloud, but using these services provides limited access to the training process and resulting models. In the interest of maintaining control over my models I wanted to avoid using cloud services.

It appeared to me that the most popular and commonly used library, at the time of taking on this project, still appeared to be TensorFlow [15]. Furthermore, this meant I would mostly be working in Python, as the core TensorFlow library is written in Python.

2.4.2 Keras API

While TensorFlow acts as a backend for ML, Keras is a high-level Python API capable of driving TensorFlow using a fairly universal set of ML concepts and ideas [16]. Keras is widely used for ML projects and is built into the TensorFlow library, making it an easy and obvious choice.

2.4.3 Matplotlib

The matplotlib library provides an API called PyPlot for producing graphs, charts and diagrams programmatically [17]. This library would come in very useful for visualising results and collected data.

3. DEVELOPING A SOLUTION

In order to develop my models, I saw two main steps that needed to be completed.

1. Producing training data for the models, generated using my existing library of music and potentially other DJs' libraries.
2. Developing and training a variety of models designed to predict the BPM from training data produced in step 1.

3.1 Producing Training Data

As a typical track is at least 2-3 minutes long and a standard waveform audio file has two stereo channels, each with a sample rate of 44,100 Hz, it simply was not feasible to use the entire raw audio data as an input variable. Instead, I needed to reduce this data down to a manageable size. I decided the best way to reduce the input was to first only select a small section of the audio. I reasoned that at least a couple seconds of audio would be needed, to ensure that multiple beats would occur during the sampled section. I justified the decision to use 2 seconds of audio as this is roughly how much audio I needed to determine the BPM of a track myself.

3.1.1 Preparing the Audio

I decided to present this short section of audio as a spectrogram. A spectrogram can convey more information than just a 2-dimensional waveform plot by applying Fourier transforms to the data. Additionally, it is possible to plot spectrograms using the mel-scale [18]. The mel-scale is used to represent the audio frequency spectrum in a subjective way based on how human hearing is able to differentiate between different frequencies.

As rhythm and musical beats are something designed to be heard by human ears, it should stand to reason that using mel-scaled spectrograms would be a more effective representation of the data for a neural network. This is something I intended to test experimentally, though.

In the interest of reducing the size of the training data I decided to use a lower sample rate and reduce the audio to a single channel (mono). I experimented with converting some audio files to lower sample rates, then listening to them to determine if I could still clearly hear the beats and determine the BPM by ear. I found that by reducing them to 22,050 Hz I could still clearly hear the beats but at 11,025 Hz I could not. As such, I settled on a sample rate of 22,050 Hz for my training data generation.

3.1.2 Interpreting a DJ's Library

I spent many hours going through my music library ensuring that all of my music was annotated as accurately as possible for this project. Typically, a DJ may have several errors in their annotations as a result of the software's tempo estimation inaccuracies. I wanted to ensure I was not introducing these errors into my dataset, hence the manual review of each track.

Using my DJ software, Rekordbox, I placed the approximately 1,400 tracks I had annotated in a playlist named "BEATNET". This totalled well over 110 hours of music that I could use. I then exported my library from Rekordbox to an XML file that would store information about the file location and tempo information of each track in my library. The exported file from my library is included in appendix B.

I wrote a Python script, *generate.py* in appendix A, to read this library file, look for songs in the BEATNET playlist, and store the location of the audio file along with tempo information. Each track in this file had a list of tempo markers. Each of these markers denoted the BPM for a portion of the track.

Some tracks only had a single marker that recorded a single BPM for the entire track, whereas others contained multiple markers. This is usually because the track changes BPM part way through.

Although, as BPMs are recorded to 2 decimal places and the start time of each marker is recorded to the nearest millisecond, sometimes a track with a BPM that cannot be represented exactly with 2 decimal places would contain multiple markers with the same BPM to account for the markers falling very slightly out of phase over time.

All of this was considered when processing the audio files, in order to produce as accurate as possible training data. This added a great deal of complexity to my data generation process and is one of the main reasons I believe my process is very effective at producing data that is accurate.

3.1.2 Spectrogram Generation

My training data generation script would select a track at random, chose a random sample of audio in each track from which to produce a spectrogram 256 pixels wide, and write it to file. My library of music was very biased towards some BPM values and so I decided to produce two datasets. The first, DS1, would cover a fairly broad range of values between 80 and 180 BPM. The second, DS2, would concentrate on a narrower range of 115-135 BPM as most of my data fell into this range. This is because the genres of music that I typically play as a DJ tend to be in this range. I reasoned that having a dataset with a more uniform distribution of data might potentially improve results.

Initially I thought that a sample length, S, of 2 seconds would be appropriate. This was because at least 2 beats would occur during the sample whilst maintaining a high resolution. However, when looking at the previous research [9], a sample of approximately 11.6 seconds was used. This was despite producing images of the same pixel dimensions.

I realised at this point that resolution might not be as important as the number of beats that occurred in each sample. For this reason, I decided to produce a variation on my two existing datasets, DS3 and DS4, that would use a longer sample length of 10 seconds for each BPM range of 80-180 and 115-135, respectively.

These four datasets were mel-spectrograms, scaled with 40 mels. As such, my spectrogram images were 40 pixels high, too. However, as I mentioned at the start of this chapter, I wanted to test if it was beneficial to use the mel-scale. I therefore produced two more datasets, DS5 and DS6, using a logarithmic scale instead. Again, with an 80-180 BPM range and a 115-135 BPM range, respectively.

3.1.3 Annotations

Initially, I intended to only produce a regression model and so the BPMs were recorded to 2 decimal places, as they are in the Rekordbox software. For example, the file name "1000-52000-124.00.png" would represent track ID 1000, starting 52,000 samples into the file with a BPM of 124. However, as this project progressed, I wanted to try using a classification approach in my experiments, too. This meant I had to slightly change the way in which I produced training data. Instead I rounded the BPM values to the nearest integer value and, in the typical fashion for a classification problem, stored each image in a sub directory named according to its label.

It might seem like rounding the BPM values to integers is a large loss of information, however, the vast majority of BPM values in electronic music are in fact integers anyway, with very few exceptions. I therefore reasoned that doing so would be an acceptable compromise and allow me to make a fairer comparison between my regression and classification models.

3.1.4 Validation and Test Sets

To perform repeatable and consistent training and evaluation of my models, it was important to pre-allocate the generated images to "training", "validation", and "test" subsets. I automated this process in my generation script by placing the images into three separate subdirectories for each subset.

Using "validation split" and "test split" parameters, I randomly assigned a fraction of the generated images to each subset. For example, a "validation split" of 0.2 would mean that 20% of the generated images would be saved to the validation subdirectory.

3.1.5 Data Augmentation

I also implemented a type of data augmentation into this script. Data augmentation involves introducing changes to the existing training data to produce even more data. It has been shown to reduce overfitting and improve performance in imbalanced datasets [19].

Directly augmenting a spectrogram image would be a complex and difficult task so I took a slightly different approach. When plotting each spectrogram, I would simply stretch the timescale by some factor and adjust the BPM annotation accordingly.

I decided that for each track I would produce a spectrogram stretched by each of a list of augmentation multipliers $\in \{0.9, 0.92, 0.94, 0.96, 0.98, 1.0, 1.02, 1.04, 1.06, 1.08, 1.1\}$.

In order to maintain precision, I multiplied the original BPM value by the multiplier and then rounded this to the nearest integer. Then, to perform the time-axis stretch, I adjusted the sample rate in my graph plotting function by a factor of $\frac{Augmented\ BPM}{Original\ BPM}$.

This strategy of stretching the time-axis of spectrograms in order to introduce data augmentation is something I also saw replicated in the previous research [9] which gave me confidence that this strategy was a good idea.

3.1.6 Additional Data

When looking at previous research I wanted to see what training data had been used. They used a combination of 3 data sets to cover various musical genres, however as my project is intended to help DJ's I was mostly interested in their electronic music dataset. This dataset [10] is publicly available and could potentially be used to extend or compare to my own dataset.

3.2 Training the Models

3.2.1 Model Architecture

As a starting point for my experiments I designed a typical basic CNN regression model architecture, shown by *Figure 1*.

My informal results from training this network using 10,000 generated spectrogram images showed that this model architecture was probably not going to be sufficient. The model appeared to simply converge on a single answer, approximately equal to the median BPM value for the training data.

The model architecture shown in *Figure 1* is a very basic CNN and has not been optimised for this task. In order to further optimise it, my first thought was to address the kernel size. As my images are spectrograms and we are most concerned with features along the time-axis when determining BPM, using a longer, thinner kernel along the time-axis might be more optimised for this

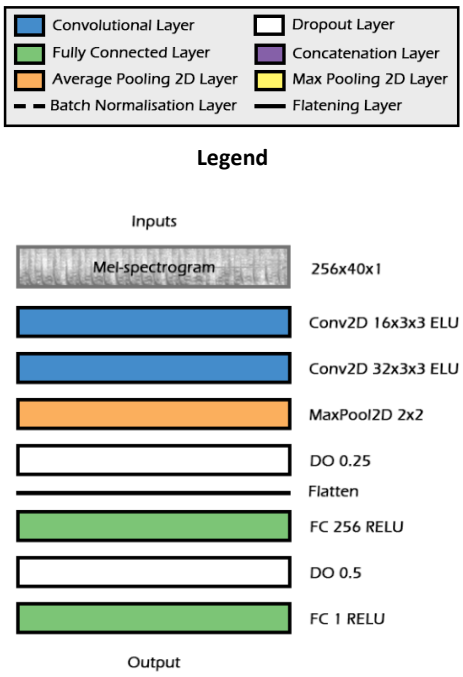

Figure 1 Initial model architecture

task. As previously discussed in section 2.2.1, this is also something that I observed in previous research [9].

Taking cues from their research, I designed another model architecture that made use of more convolutional layers; with longer, thinner kernels oriented along the time axis. First of all, I started with a still relatively simple shallow model architecture. Secondly, I tried to replicate their model architecture more closely to produce a deep architecture, making use of the multi-filter modules they proposed. I will refer to these two architectures as shallow and deep, respectively.

Additionally, as I noted in section 2.2.1, this research uses a classification approach to the problem, despite a regression approach perhaps being more intuitive. As such, I produced 2 variations of each model architecture. A regression type, with a single fully connected node as the last layer and an ELU activation, and a classification type, with N=(number of classes) fully connected nodes and a softmax activation function.

In summary, I had developed 4 different network architectures, shown by *Figure 2* and *Figure 3*. I wanted to train each of these on each of my 6 datasets to produce a total of 24 experimental models. Once trained, I would be able to compare them to each other and perhaps to existing algorithms to determine the success of each model and the project as a whole.

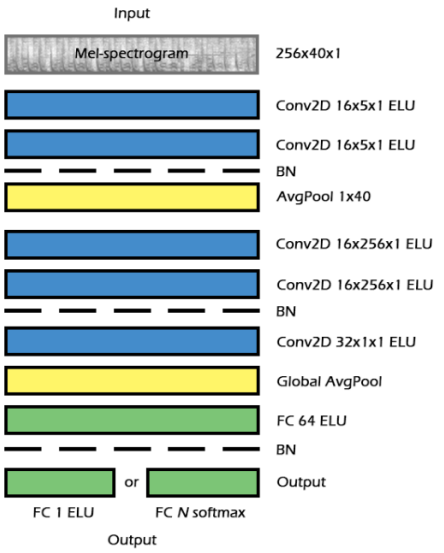

Figure 2 Shallow model architecture

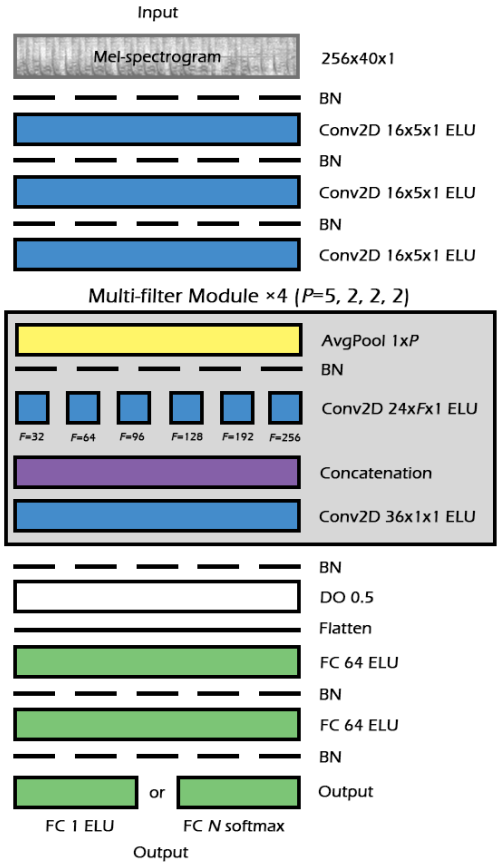

Figure 3 Deep model architecture

3.2.2 Training and Evaluating Models

In informal experiments, I found the models usually reached their best weights within approximately 10-15 epochs using a batch size of 128 and 100 steps per epoch. In my formal experiments, I would run the models for up to 100 epochs using the same batch size and steps but with an early stopping condition. If the validation loss did not improve for 3 continuous epochs, the model would stop training.

Both the final iteration of each model, as well as the best weights, would be saved as *model_name.final.h5* and *model_name.best.h5*, respectively. These can be found in appendix D. For

my final model evaluation, I would test the best weights model against a separate, previously unseen test set of images.

In order to train each model I produced the *train.py* script found in my source code in appendix A. This would take as parameters: a dataset; a model architecture (deep or shallow); and an output mode (classification or regression) to produce a trained model as detailed above.

Also in the source code is *evaluate.py* that, given a model, will make predicitons using the relevant test set. Metrics to evaluate the models performance such as MSE and accuracy were calculated and logged to a results spreadsheet for later. Additionally, a full log of every prediction made was logged to a separate spreadsheet for each model, to allow for further investigation of results.

To automate the procedure of training and evaluating all 24 models, I produced the *train_eval.py* script that would execute the *train.py* and *evaluate.py* scripts for each unique set of parameters.

Lastly, in order to use the model as it was intended, I produced the *estimate.py* script that accepts either a spectrogram or an audio file as input. Given an audio file, a random spectrogram is generated to use as the model input. This could have been improved to sample several sections of the same track and produce an averaged estimate, but as the final models were not particularly successful, this was not something I felt was necessary to implement.

4. RESULTS AND EVALUATION

4.1 Model Evaluation Methods

4.1.1 Traditional Metrics

As I have a combination of both classification and regression models to compare, some thought needed to be placed into methods of model evaluation, as I was unable to find any standard methods of comparison.

Typically, a regression model might be evaluated by looking at its mean-square-error (MSE) and mean-absolute-error (MAE). Indeed, I did use MSE as the loss function to train my models. However, MSE is less useful for understanding the contextual performance of the model. The MAE may still be a useful figure, as this will tell us how many BPM an average estimation is incorrect by. This may be heavily skewed by outliers though, so it is important to consider other metrics.

With classification models, performance is typically evaluated by looking at categorical cross-entropy (CCE) and accuracy. CCE was the chosen loss function when training my models, as is common practice. With that said, since my categories do represent numerical value, it is possible a refined loss function could have improved results. Additionally, in this instance, we use the term accuracy to mean an exact, correct prediction.

Due to the nature of tempo estimation being a numerical result, it is possible to convert the classification model predictions to numerical values and calculate MSE and MAE. This is not usually the best method for evaluating classification models, however for the purpose of this project, it does provide a method of comparison between the classification and regression models.

The method of converting a categorical result into a numeric value is typically to take the model output with the highest value to be the prediction, often referred to as the "argmax". It could also be possible to use an equation such as that proposed in an age estimation model [20] that takes into account the values of all model outputs, summing each output multiplied by some factor in order to produce a more nuanced estimate. I decided not to take this approach, however, as the results will show that

the models were not very successful anyway. As such, refining these estimations further would not produce much benefit.

4.1.2 Tempo Estimation Accuracy Metrics

As discussed in section 2.2.2 though, there are two standard metrics for evaluating tempo estimation algorithms, as previously discussed in section 2.2.2. The first metric, referred to as accuracy1, considers an estimation to be accurate if it falls within ±4% of the true value. The second metric, accuracy2, considers an estimation to be accurate if it falls within ±4% of the true value, or the true value multiplied by one of $\frac{1}{3}$, $\frac{1}{2}$, 2, or 3. This takes into account the so called "octave errors" and shows the difference between exact estimations, and those that fall into this caveat.

I therefore calculated these two metrics for all my models as my main point of comparison. Additionally, this allows me to compare my model's performance to that of other algorithms as detailed in *Table 1*.

4.2 Results

Dataset	Model Architecture	Model Type	BPM Range	MSE	MAE	Accuracy1	Accuracy2
DS1	Deep	Classification	80-180	105.00	8.00	38%	38%
DS1	Deep	Regression	80-180	910.66	14.21	33%	33%
DS1	Shallow	Classification	80-180	129.00	9.00	34%	34%
DS1	Shallow	Regression	80-180	307.37	12.83	25%	26%
DS2	Deep	Classification	115-135	64.00	6.00	45%	45%
DS2	Deep	Regression	115-135	578.63	22.73	0%	0%
DS2	Shallow	Classification	115-135	71.00	6.00	43%	43%
DS2	Shallow	Regression	115-135	37.33	5.24	46%	46%
DS3	Deep	Classification	80-180	267.00	12.00	30%	30%
DS3	Deep	Regression	80-180	1,908.14	33.58	2%	2%
DS3	Shallow	Classification	80-180	273.00	11.00	30%	30%
DS3	Shallow	Regression	80-180	217.62	10.70	32%	32%
DS4	Deep	Classification	115-135	82.00	7.00	38%	38%
DS4	Deep	Regression	115-135	1,178.11	19.42	16%	16%
DS4	Shallow	Classification	115-135	80.00	7.00	39%	39%
DS4	Shallow	Regression	115-135	56.55	6.19	47%	47%
DS5	Deep	Classification	80-180	218.00	11.00	31%	31%
DS5	Deep	Regression	80-180	1,660.36	28.64	6%	6%
DS5	Shallow	Classification	80-180	203.00	10.00	34%	34%
DS5	Shallow	Regression	80-180	273.47	13.63	23%	23%
DS6	Deep	Classification	115-135	79.00	7.00	40%	40%
DS6	Deep	Regression	115-135	1,850.64	19.22	44%	44%
DS6	Shallow	Classification	115-135	52.00	5.00	46%	46%
DS6	Shallow	Regression	115-135	56.11	5.63	49%	49%

Table 2 Model evaluation results.

Training graphs for each model can be found in appendix C. The trained models themselves can be found in appendix D.

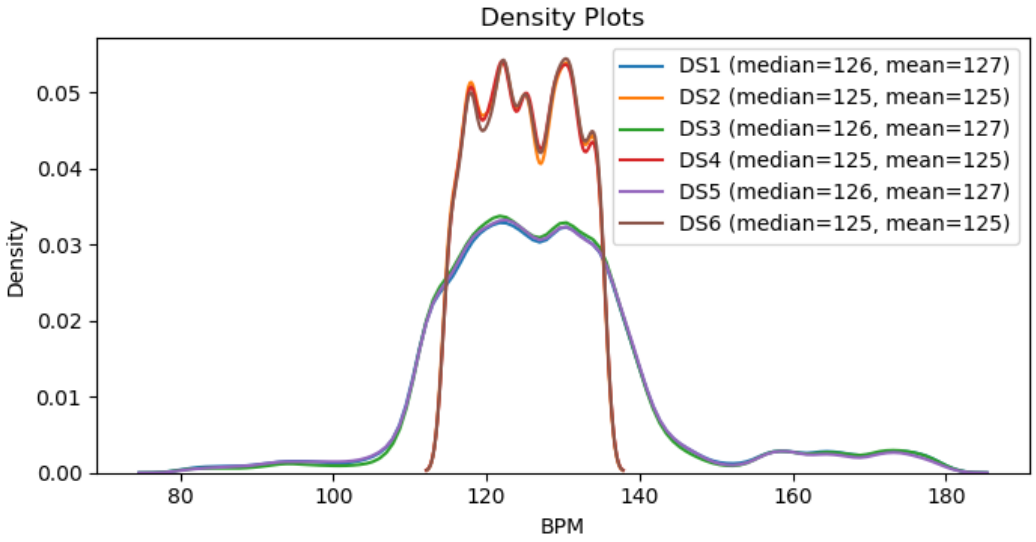

Figure 4 Density plots of each dataset, labelled with median and mean values.

4.3 Evaluation of Results

4.3.1 Accuracy Evaluation

By looking at the accuracy1 measurements, we can see that some of the models appear to be performing somewhat well. Looking at the accuracy2 measurements, we can see that these models are not strongly affected by the "octave errors", as there is very little difference between accuracy1 and accuracy2 for all models. However, there are several problems with this evaluation.

Firstly, half of my datasets use a very limited BPM range of only 20. Due to this limited range, we would expect to see significantly better results than the models trained over broader datasets. Especially when we consider that ±4% from the midpoint of this range would encompass 50% of the possible predicted values for these datasets. For this reason, these accuracy measurements appear misleadingly high for these models and are in fact, very poor.

Secondly, when considering my broader datasets, we see that these accuracy figures do not fall as sharply as might be expected, considering the much wider range. This may suggest that these models are perhaps performing better in general. However, when considering the distribution of the datasets, shown in *Figure 4*, this is not necessarily the case. A majority of the tracks fall into a relatively small range only a few percent either-side of the median of each dataset.

Tables of each models' predictions for the test set can be found in appendix E. When we look at these predictions, we can see the models are in fact performing very poorly. Anecdotally, in general, the models seem to have a small number of common predictions, with the median value being very common. As such, the accuracy1 and accuracy2 metrics used above are not very good at evaluating these particular models, due to the small BPM range and distribution of training data.

4.3.2 Comparing to Alternative Algorithms

By looking at the accuracy results in table 1 for alternative algorithms and software, especially when taking into account the range and distribution issues described above, we can see that the performance of my models is worse than any existing solutions.

There are perhaps two potential exceptions to this. First of all, the [21] algorithm is outperformed on accuracy1 by some of my models. However, the accuracy2 score is no better than accuracy1, unlike

their results, which would make their algorithm more useful in a practical setting. Secondly, this algorithm could be applied to music of almost any BPM due to its methodology involving detecting individual beat onsets and deriving the tempo.

Secondly, the values taken from the *Beatport* website are drastically lower than all of the other algorithms, including mine. However, when we consider the fact that the dataset used to benchmark these algorithms was constructed from user corrections submitted to their website, there is understandably a very strong bias acting against that particular algorithm.

4.4 Evaluation of Methodology

The question remains, is machine learning a useful tool for tempo estimation of electronic music? The previous research already discussed in this dissertation would suggest there is a potential. Unfortunately, I was not able to replicate the same success. However, I believe this to be down to a combination of my poorly distributed training data, as well as my own inexperience in the field of machine learning.

This project was a learning experience for me, having very little prior experience with the technologies involved. As such, I was very reliant of previous work of others to guide my approach. I do believe that with a greater level of background knowledge and experience I, or others alike, could be able to produce a more viable tempo estimation algorithm using machine learning.

Whether the single-step approach used in this project, and by Schreiber and Muller, is the right way to approach the problem, I am not entirely sure. I think further investigation is needed. There is also the potential that developing an algorithm centred around beat onset detection and deriving the tempo, similar to the other non-machine learning algorithms discussed, may be more successful. Additionally, most commercial DJ software applications do not disclose their methodology of tempo estimation, as this is often a key selling point of the software. I think it is likely that at least some of these software packages are using some degree of machine learning.

5. CONCLUSIONS

5.1 Hypothesis Review

The hypothesis proposed in this project was:

> *Existing tempo estimation algorithms could be improved using machine learning algorithms trained on a DJ's own music library.*

In the end, I was not able to conclusively prove my hypothesis to be true. However, I do not think I have conclusively shown otherwise, either. As such, further research is needed. I do believe there is promise in this approach to tempo estimation, as the problem of tempo estimation is one that should be well suited to ML.

While I was unable to produce successful models in my own work, there is evidence to suggest it could be possible. Perhaps by combining other researchers more successful models with my novel approach to data collection, a better algorithm could be developed yet.

5.2 Objectives Review

5.2.1 Collect and label a large enough dataset of music samples

For this project I annotated 1,379 individual tracks from my own music library, totalling over 114 hours of audio. This is significantly larger than any publicly available dataset of electronic music. The

GiantSteps Tempo Dataset discussed in section 2.2.2 includes just 664 tracks, of which 1-minute samples are available publicly for a total of just over 11 hours of audio.

Additionally, that dataset assumes a single BPM value for each track which is not always correct. My training data generation process is able to account for varying tempos during a track by interpreting a Rekordbox Collection export, which can have multiple tempo annotations per track.

The only caveat of my process was that my own library of music perhaps did not offer enough variety of music. As shown by *Figure 4*, my dataset contains a strong bias towards certain tempos caused by my preference for certain genres of music.

However, the benefit of my approach is that it can be very easily replicated by other DJs using my data generation script. With some improvements to the general usability of this script, I believe it could be used to collect a *very* significant and broad dataset for future research.

5.2.2 Investigate methods of pre-processing the audio

In my experiments, I produced spectrograms using various parameters such as varying the sample length and y-axis scale. Unfortunately, as my models were not very successful, it was difficult to draw any particular conclusions about the success of these various methods. I do still feel I made a good attempt at investigating various methods, though.

I could have expanded upon this by investigating approaches involving pre-processing of the audio signal, for example using a low-pass filter. Or, I could have represented the audio as a different type of plot, for example a waveform-plot or a chromogram.

5.2.3 Research different types of neural network and their uses

During this project I learned a lot about different types of neural network. I decided fairly early on that I would place my focus on using a CNN as this is one of the most commonly used types of neural network for such a task. However, if I had more time on this project, I could have perhaps investigated a broader range of neural networks.

For example, a recurrent neural network (RNN) is one that is able to process variable length sequences of inputs by taking advantage of an internal state (memory). Perhaps this style of model would be useful for the task at hand, as musical tracks are themselves variable length sequences. Some RNNs make use of a separate class of ANN called long short-term memory networks (LSTMs) as their internal state, that could also be applicable in this instance.

5.2.4 Train and evaluate my own neural network models

Over the course of this project I feel that I developed a comprehensive and effective method for training and evaluating various different models. This can be seen by reviewing my source code in appendix A. Using my source code, it would be quite easy to conduct further experiments with different model architectures and datasets, with much of this process being automated. I do think that my source code could perhaps be refactored and improved, though, in order to further generalise it and make this process a little simpler.

5.2.5 Compare the accuracy of my models to existing methods

Using the standard accuracy1 and accuracy2 metrics first described in section 2.2.2 of this document, I was able to compare my models' performance to that of both previous research and industry software. This comparison was not perfect, as my approach to estimation involved a much more limited range of values than most algorithms. However, it was still able to show that my algorithms were not as effective as others, which was ultimately the goal of this objective.

5.3 Learning Review

Before embarking on this project, I had very little knowledge or experience on the topic of machine learning algorithms. I thought of them as almost mystery black-box algorithms that were capable of almost any easily defined and repeatable task. By the end, I felt I had learned a great deal about them. But mostly I learned just how broad and complex the area is; and just how much more there was still to learn.

I learned that the process of designing model architecture is extremely important in ensuring the success of an ML algorithm. The models require a great deal of guidance and structure in order to effectively set them up for success, there is not just a single generalised approach to designing them.

I was surprised by how much mathematics I needed to understand in order to effectively make use of ML algorithms in new and novel ways. I expected that by using machine learning algorithms, I would not have to learn as much about the underlying problem at hand, but in fact the opposite was true. I think that in order to make effective use ML, a very good understanding of the problem is needed in order to produce a good design.

I'm more inspired and fascinated by the topic of ML than I was going into this project and I'm excited to continue working on it into the future, as well as to continue to experiment with alternative applications.

5.4 Future Work

Obviously, further work is needed on this project in order to conclusively prove or disprove my hypothesis. I'm not sure if my models failed due to their design, perhaps I have misinterpreted and not properly implemented the more successful designs proposed in previous works. Or, perhaps they failed due to differences in my training data.

In order to find out and determine the best course for future action, I think I need to train my models using the GiantSteps Tempo Dataset used in previous works. This might indicate if my training data was the problem. Additionally, it could be interesting to perform experiments using controlled subsets of my training data, to investigate how training set size influences the quality of results.

While my training data was not perfect, I do believe my methods of collecting it provide a very strong basis for future work. There are countless DJ's in the world with extensive music libraries, many of which are already at least partially annotated. With some work, I believe that a very comprehensive dataset could be produced by taking advantage of this fact. I intend to contact the authors of previous works in this area of research in order to discuss the possibility of this.

REFERENCES

[1] A. L. Samuel, "Some studies in machine learning using the game of Checkers," *IBM Journal of Research and Development,* pp. 71-105, 1959.

[2] S. J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed., Patience Hall, 2009.

[3] T. M. Mitchell, Machine Learning, McGraw-Hill, 1997.

[4] S. Kleene, "Representation of Events in Nerve Nets and Finite Automata," *Annals of Mathematics Studies,* vol. 34, pp. 3-41, 1956.

[5] K. Fukushima, "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," *Biological Cybernetics,* vol. 36, pp. 193-202, 1980.

[6] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *The Journal of Physiology,* vol. 195, no. 1, pp. 215-243, 1968.

[7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* vol. 15, no. 56, pp. 1929-1958, 2014.

[8] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Lille, France, 2015.

[9] H. Schreiber and M. Muller, "A Single-Step Approach To Musical Tempo Estimation Using a Convolutional Neural Network," Paris, France, 2018.

[10] P. Knees, Á. Faraldo, P. Herrera, R. Vogl, S. Böck, F. Hörschläger and M. L. Goff, "Two Data Sets for Tempo Estimation and Key Detection in Electronic Dance Music Annotated from User Corrections," Málaga, Spain, 2015.

[11] F. Patin, "Beat Detection Algorithms," 2003. [Online]. Available: http://archive.gamedev.net/archive/reference/articles/article1952.html. [Accessed 12 02 2020].

[12] J. Sullivan, "Beat Detection Using JavaScript and the Web Audio API," [Online]. Available: http://joesul.li/van/beat-detection-using-web-audio/. [Accessed 20 02 2020].

[13] G. Tzanetakis, G. Essl and P. Cook, "Audio Analysis using the Discrete Wavelet Transform," Wisconsin, United States, 2001.

[14] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia and L, "TensorFlow: Large-scale machine learning on heterogeneous systems,," 2015. [Online]. Available: tensorflow.org. [Accessed 12 03 2020].

[15] S. Martin, "10 Most Popular Machine Learning Software Tools in 2020 (updated)," 27 08 2019. [Online]. Available: https://towardsdatascience.com/10-most-popular-machine-learning-software-tools-in-2019-678b80643ceb.

[16] F. Chollet, "Keras," 2015. [Online]. Available: keras.io.

[17] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering,* vol. 9, no. 3, pp. 90-95, 2007.

[18] S. S. Stevens, J. Volkmann and E. B. Newman, "A Scale for the Measurement of the Psychological Magnitude Pitch," *Journal of the Acoustical Society of America,* vol. 8, no. 3, pp. 185-190, 1937.

[19] S. C. Wong, A. Gatt, V. Stamatescu and M. D. McDonnell, "Understanding Data Augmentation for Classification: When to Warp?," Gold Coast, QLD, Australia, 2016.

[20] R. Rothe, R. Timofte and L. Van Gool, "DEX: Deep EXpectation of apparent age from a single image," Long Beach, California, 2015.

[21] M. E. P. Davies and M. D. Plumbley, "Context-Dependent Beat Tracking of Musical Audio," *IEEE Transactions On Audio, Speech, and Language Processing,* vol. 15, no. 3, pp. 1009-1020, 2007.

APPENDIX

Appendix A: Source Code

The final source code for this project can be found in a directory named "Appendix A: Source Code" in the supplementary materials provided with this document. Additionally, it is hosted publicly on *GitHub*[3] where it will continue to be updated.

```
Appendix A: Source Code/
├── .gitignore
├── density_plot.py
├── estimate.py
├── evaluate.py
├── generate.py
├── LICENSE
├── README.md
├── requirements.txt
├── train.py
└── train_eval.py
```

Appendix B: Rekordbox Collection

The exported Rekordbox Collection in xml format can be found in a directory named "Appendix B: Rekordbox Collection" in the supplementary materials provided with this document.

```
Appendix B: Rekordbox Collection/
└── lib.xml
```

[3] https://github.com/djwillcaine/BeatNet

Appendix C: Training Graphs

The training graphs produced during training can be found in a directory named "Appendix C: Training Graphs" in the supplementary materials provided with this document.

```
Appendix C: Training Graphs/
├── DS1.deep.classification.80-180.png
├── DS1.deep.regression.80-180.png
├── DS1.shallow.classification.80-180.png
├── DS1.shallow.regression.80-180.png
├── DS2.deep.classification.115-135.png
│    ...
└── DS6.shallow.regression.115-135.png
```

Appendix D: Models

The resulting "best" models trained for this project can be found in a directory named "Appendix D: Models" in the supplementary materials provided with this document. The "final" models have been excluded due to file size restrictions.

```
Appendix D: Models/
├── DS1.deep.classification.80-180.best.h5
├── DS1.deep.regression.80-180.best.h5
├── DS1.shallow.classification.80-180.best.h5
├── DS1.shallow.regression.80-180.best.h5
├── DS2.deep.classification.115-135.best.h5
│    ...
└── DS6.shallow.regression.115-135.best.h5
```

Appendix E: Model Test Results

A verbose log of test predictions for each model produced can be found in a directory named "Appendix D: Model Test Results" in the supplementary materials provided with this document.

```
Appendix E: Model Test Results/
├── DS1.deep.classification.80-180.csv
├── DS1.deep.regression.80-180.csv
├── DS1.shallow.classification.80-180.csv
├── DS1.shallow.regression.80-180.csv
├── DS2.deep.classification.115-135.csv
│    ...
└── DS6.shallow.regression.115-135.csv
```