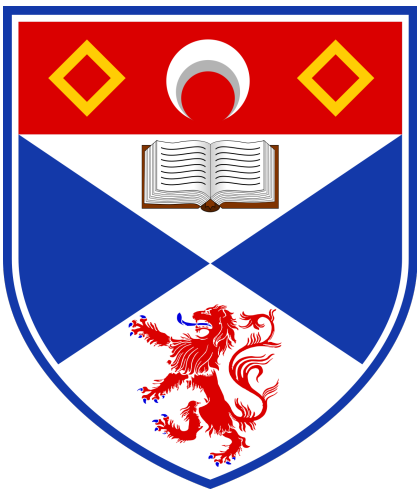
Automatic Polyphonic Guitar Transcription

Vanaj Krishna Moorthy

200006295



Project Supervisor: Dr Ognjen Arandjelović

School of Computer Science
University of St Andrews

1st of April 2024

Contents

***Abstract*—This paper introduces "PolyTab," a robust machine-learning model leveraging Convolutional Neural Networks (CNNs) for the automated transcription of polyphonic guitar audio into tablature. The model innovatively applies a learnable weighted loss function and the AdamW optimiser, which demonstrably enhances its performance over traditional transcription methodologies. "PolyTab" excels in its predictive accuracy of string-fret combinations, as evidenced by rigorous metric evaluations. Additionally, the PolyTabPredictor class offers a real-time transcription tool, incorporating a novel temporal smoothing algorithm to ensure the playability of generated tabs. Comparative analyses against existing models reveal "PolyTab's" promising advancements in the field, despite challenges in generalisability and computational efficiency during training.**

I. Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main text of this project report is 8,383 words long, including project specification and plan. In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web. I retain the copyright in this work."

II. Introduction

Guitar tablature is the most prevalent and pervasive form of music notation in the modern era. Owing to the guitar's popularity amongst both professionals and amateurs, and the accessibility provided by the internet, traditional sheet music has slowly but surely been phased out by modern guitarists for more accessible tablature notation instead [1]. Until recently, however, the task of transcribing music to notation has relied solely on arduous manual transcription performed by humans. Automatic guitar transcription attempts to generate a symbolic representation of the pitches played in the audio in order to guide guitarists trying to learn the piece. The project detailed within this paper focuses on the automated generation of guitar tablatures from audio recordings.

My primary objective was to have a working machine-learning model that could transcribe polyphonic isolated guitar recordings into guitar tabs with some accuracy. At the core of this endeavour is the recognition of the nuanced relationship between audio signals and their corresponding musical notations. The development of my model was guided by the dual objectives of accuracy in musical transcription and the practical applicability of the generated tabs. To achieve these goals, I leveraged Convolutional Neural Networks (CNNs) to craft a system capable of analysing audio recordings and outputting detailed guitar tabs. This process not only includes the identification of notes and chords but also their precise positioning on the guitar's fretboard, a task that presents significant challenges due to the instrument's polyphonic nature and the potential for multiple fingering options for the same chord.

By automating the tablature generation process, I help move towards a more accessible means of musical transcription, potentially democratising the learning process for budding musicians. Moreover, this project contributes to the broader field of Music Information Retrieval (MIR), showcasing the potential of deep-learning techniques in interpreting and transcribing complex audio signals.

This paper is structured to provide a comprehensive overview of my project, beginning with a review of relevant literature in the fields of automatic music transcription and guitar tablature generation. I then detail the methodology behind the development of our deep-learning model, including data preparation, architectural decisions, and training processes. Following this, I present the results of our model's performance, evaluated against a diverse dataset of music spanning various genres and complexities. The discussion that follows reflects on the implications of our findings, potential applications, and avenues for future research.

III. Context Survey

Automatic guitar tab transcription is a challenging task due to the intricacies of translating audio signals, musical notes, and finger placements on the fretboard. Researchers have explored various approaches over the years, with significant advancements in recent times thanks to deep-learning.

A. Traditional Approaches

Early efforts in automatic guitar transcription relied on techniques that predate deep-learning.

Rule-based Systems: These systems implement a set of pre-defined rules based on music theory and audio signal processing knowledge [1]. They analyse features like pitch, amplitude, and note duration to infer tablature. While offering interpretability, rule-based systems struggle with the inherent ambiguity in guitar music, especially when dealing with techniques like bends, slides, and effects [2]. Additionally, the complexities of polyphony (multiple notes played simultaneously) pose challenges for these rule-based approaches.

Hidden Markov Models (HMMs): HMMs are statistical models that capture the sequential nature of musical data [3]. Trained on labelled guitar recordings, HMMs predict the most likely sequence of notes based on observed audio features. However, HMMs can be computationally expensive and may struggle with the complexities of polyphonic guitar music [4].

Non-negative Matrix Factorization (NMF): NMF decom-

poses a spectrogram (a visual representation of an audio signal's frequency content over time) into a set of basis vectors and corresponding activation coefficients [5]. These basis vectors can potentially represent instruments or individual notes. However, separating guitar components from a mix (where other instruments are present) and accurately linking them to specific notes on the fretboard remains a challenge [6].

B. Deep-Learning Approaches

The emergence of deep-learning has revolutionised automatic guitar transcription in recent years. Convolutional Neural Networks (CNNs) have proven particularly effective due to their ability to learn complex patterns directly from audio data.

TabCNN: A foundational work in this area is TabCNN by Wiggins et al. (2019) [7]. This paper proposes a CNN architecture specifically designed for guitar tablature prediction. It takes a Constant-Q Transform (CQT) representation of the audio signal as input and utilises convolutional layers to extract features related to pitch and time. The final layers predict the fret positions for each string at each time step. TabCNN demonstrated significant improvements in accuracy compared to traditional approaches.

Beyond TabCNN: Following TabCNN's success, researchers have actively built upon this work. Variations in CNN architectures have been explored, experimenting with different filter sizes, pooling layers, and activation functions (e.g., [8, 9]). My project similarly attempts to build upon TabCNN's approach by experimenting with the loss function and optimisation algorithms to produce a more refined model.

IV. Software Engineering Process and Technology Decisions

I wrote this project in the Python programming language as it is widely recognised as a leading programming language in the field of machine-learning and data science, thanks to its simplicity and readability, alongside a robust ecosystem of libraries and frameworks that streamline development. This makes Python an ideal choice for tasks that involve data manipulation, scientific computing, and machine-learning. I also used some standard Python libraries based on the respective strengths and the functionalities they offer:

- NumPy: A fundamental package for scientific computing with Python, NumPy offers support for large, multi-dimensional arrays and matrices, alongside a collection of high-level mathematical functions to operate on these arrays. NumPy is heavily used for numerical computations, which form the backbone of data processing in machine-learning.

- SciPy: Built on NumPy, SciPy extends its capabilities by adding useful features for optimisation, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, and other tasks common in science and engineering.

- Pandas: An indispensable tool for data manipulation and analysis, Pandas provides high-performance, easy-to-use data structures. It is particularly suited for handling structured data and performing complex data manipulation and querying.

- JAMS: The JSON Annotated Music Specification (JAMS) is a package designed for music information retrieval tasks. It provides a standardised format to store annotations and metadata of music tracks, which is essential for processing and interpreting the .jams files that accompany my audio files, ensuring structured and consistent metadata management.

- Librosa: A Python package for music and audio analysis, Librosa provides the building blocks necessary to create music information retrieval systems. With functionalities for audio signal processing such as feature extraction, it's a go-to library for tasks like audio analysis, beat tracking, or extracting spectrograms, making it ideal for pre-processing audio data for machine-learning models.

- Keras: An open-source software library that provides a Python interface for artificial neural networks, Keras acts as an interface for the TensorFlow library. It's known for its user-friendliness, modularity, and extensibility, allowing for easy and fast prototyping of deep-learning models.

- TensorFlow: TensorFlow is one of the most popular machine-learning libraries. It provides a comprehensive, flexible ecosystem of tools, libraries, and community resources that allows developers to easily build and deploy ML-powered applications.

- Matplotlib: A plotting library for the Python programming language. It provides an object-oriented API for generating plots, which is vital for visualising data and results, and I am also using it to generate the final tablatures my model outputs.

My decision to use these libraries is rooted in the need for robust, scalable, and efficient tools to handle the demands of processing large datasets, extracting meaningful features from complex audio signals, and constructing sophisticated machine-learning models. TensorFlow, paired with Keras, forms a powerful duo for deep-learning, offering both the granular control needed for research and experimentation, and the simplicity required for rapid prototyping. The use of Python and these libraries reflects industry best practices, ensuring that my work is built upon reliable and well-supported tools that are both flexible and performance-oriented.

V. Ethics

My project required no ethics approval. All the data used for training the model was open source and no user survey was conducted. The signed ethics approval form can be found in Apendix A.

VI. Dataset

The dataset I have employed for this project is the GuitarSet dataset [2]. GuitarSet is a comprehensive dataset designed to foster research in guitar transcription and related Music Information Retrieval (MIR) tasks. It consists of 360 audio recordings of solo acoustic guitar performances, each approximately 30 seconds in length. These recordings cover a variety of keys and musical styles including bossa nova, rock, and funk, and also span every key. Each performance in the dataset is uniquely captured using a guitar equipped with a hexaphonic pickup, allowing for the isolation and separate processing of the signals from each individual guitar string. This setup facilitates the generation of frame-level pitch annotations for each of the six strings, making it possible to access the ground truth fretting utilised by the guitarist. While this hexaphonic setup is utilised for generating the annotations, my model uses the single microphone monophonic recordings to train on. The dataset notably includes two distinct renditions for each piece: one focusing on soloing (primarily single notes) and another on comping (chord playing), performed by six different guitarists. This diversity encourages a comprehensive analysis of guitar playing techniques and styles.

VII. Audio Preprocessing

My machine-learning model employs an audio preprocessing pipeline designed to transform raw guitar audio into a format conducive for Convolutional Neural Network (CNN) analysis. This preprocessing pipeline is a crucial step, laying the foundation for accurate and efficient guitar tablature transcription. The methodology encompasses several key processes: downsampling, normalisation, and the generation of Constant-Q Transformations (CQTs). Each of these steps is tailored to ensure that the input data maximises the CNN's ability to discern and classify fret-string positions accurately during guitar performances.

A. Downsampling

Downsampling is the initial step in the preprocessing phase, where the original audio signal's sampling rate is reduced. This process is essential for two primary reasons: it diminishes the computational load by reducing the amount of data to be processed and helps focus the model on the most relevant frequencies by eliminating higher-frequency content that is less significant for the task at hand. In my model, audio signals are downsampled from 44,100 Hz, the frequency the audio was recorded at, to 22,050 Hz. Following the Nyquist-Shannon theorem, this allows us to use frequencies up to 11,025 Hz [3]. This covers most of the harmonic range produced by the guitar and allows us to reduce the dimensionality of the input signal and is also a common practice that balances computational efficiency with the preservation of audio quality [4].

B. Normalisation

Following downsampling, normalisation is applied to the audio signals. This step ensures that all inputs into the model have a consistent amplitude range, preventing any particular recording's volume from disproportionately influencing the model's performance. Normalisation is achieved by scaling the amplitude of all audio files so that their peak amplitude matches a set target, thus standardising the dynamic range across the dataset. In Fig 1. we can see the audio file after downsampling and normalisation. The downsampling is not visually very apparent as the waveform looks similarly dense, but this helps convince us that not too much relevant information is lost by downsampling. We can also easily see the amplitude being normalised between 1 and -1 by observing the y-axis.

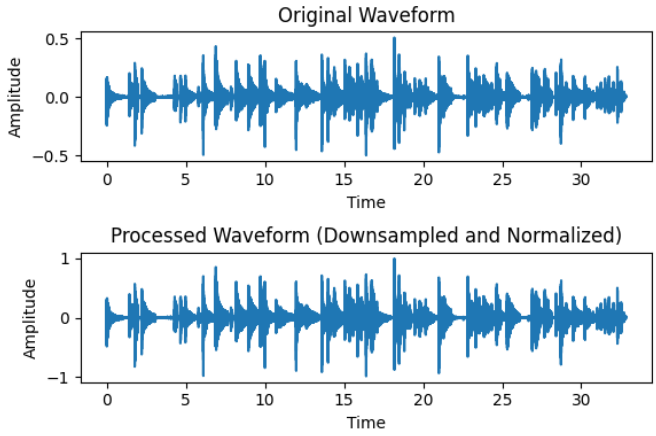


Figure 1: Comparison of the original audio waveform before and after normalisation and downsampling.

VIII. Generating Constant-Q-Transforms

Due to CNNs being geared towards learning spatial features from images, I transformed my audio signals into image representations. Motivated by previous work [5], I decided to convert my audio signals into Constant-Q-Transform representations. CQT is an audio signal processing technique that provides a time-frequency representation of a signal, where the frequency bins are geometrically spaced, and each bin has a constant Q factor (quality factor). The Q factor is a measure of the frequency bandwidth relative to the centre frequency, allowing for a more detailed low-frequency analysis and a coarser high-frequency analysis. This is particularly advantageous for music processing, as it aligns more closely with the human auditory system's perception and the musical scales' logarithmic nature.

The Short-time Fourier transform (STFT) is another common image representation that could be used for this task. While both CQTs and STFTs provide time-frequency representations, there are key differences that make CQTs more suitable for musical signal analysis. STFTs have constant frequency resolution across all frequencies, which might not capture low-frequency details well. CQTs offer higher resolution at lower frequencies, making them better for analysing musical notes that often lie in this range. The logarithmic spacing of frequency bins in CQTs also mirror the human auditory system's logarithmic frequency perception and the musical scale. This makes CQTs more aligned with how humans perceive music, offering more

intuitive and musically relevant features for analysis. The geometric spacing of frequency bins in CQTs also ensure that the harmonics of musical notes align vertically in the spectrogram. This property is beneficial for tasks like chord detection, pitch tracking, and timbre analysis, where harmonic relationships are crucial.

Based on previous work, I decided to use a CQT with 192 bins across 8 octaves [5], this parameter specifies the total number of frequency bins in the CQT, directly impacting the frequency resolution. A higher number of bins allows for finer granularity in frequency analysis, essential for distinguishing between closely spaced musical notes.

192 bins gives us 24 bins per octave, or 2 bins per semitone. This influences the detail level captured within each octave. With 24 bins per octave, the model can accurately capture semitone differences, aligning well with Western music's twelve-tone equal temperament system.

I also use a hopsize of 512 samples, which equates to 43 frames per second. This determines the stride with which the analysis window moves across the audio signal, affecting the time resolution of the resulting CQT. A smaller hop length results in higher temporal resolution, allowing for more precise localisation of musical events in time.

Finally, to incorporate temporal context into the model, a sliding window approach is adopted, where 9 consecutive frames are considered together and are padded on either size to make sure the input shape is uniform. This method enables the model to leverage information from adjacent frames, improving its ability to capture temporal dynamics and relationships between successive musical notes. As a result, this gives us input samples 192 x 9 at each frame.

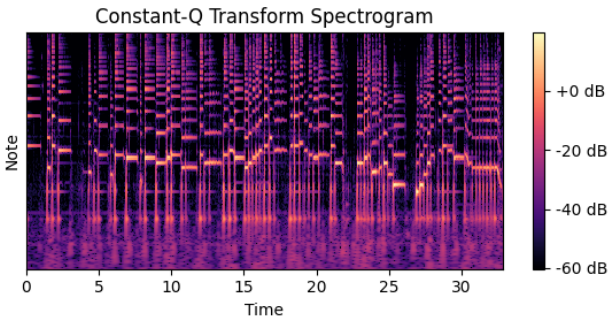


Figure 2: An example CQT spectogram generated from one of the entire audio files.

IX. Label Preprocessing

GuitarSet also comes with annotations for each of the audio files in the JAMS file format [2]. I extract these annotations and process them to use as labels for my model.

A. Extraction of Annotation

The annotations for each guitar performance are stored in .jams files. These annotations include detailed information about the played notes, including their onset times, durations, and MIDI note numbers. My preprocessing script begins by loading these .jams annotations for each corresponding audio file. This step ensures that there is a direct mapping between the audio being analysed and the musical information provided by the annotations.

B. Mapping MIDI notes to Guitar Fret-String combinations

One of the core tasks of the label preprocessing is to convert MIDI note numbers into specific fret-string combinations on the guitar. This conversion is essential because my model aims to predict these physical playing positions rather than abstract musical notes. The conversion process takes into account the standard tuning of the guitar strings and the playable range of frets on the instrument. I sample the stringwise pitch annotations at the same rate of 43 frames per second. I round each sampled MIDI pitch to the nearest whole number and depending on which string it was played on, I subtract the value of the same open string. As a result, I get an integer value for the fret number played on each string with a value of 0 indicating. There are 19 frets I am considering since this is standard for acoustic guitars. Since a string can be played open and can also not be played, this gives me 21 different fret classes that each string may be in during a frame.

C. Temporal Alignment with CQT Frames

The annotations need to be temporally aligned with the CQT frames generated from the audio signal. This alignment ensures that each frame of input features has a corresponding label that indicates the fret-string positions being played at that moment in the audio. The alignment process involves calculating the time each CQT frame represents and then determining which notes (and therefore which fret-string positions) are active during each frame. This temporal alignment is crucial for training the model to associate specific audio features with the correct playing positions.

D. Encoding Labels for Model Consumption

Once the fret-string combinations have been determined for each frame, I one-hot encode these labels [6]. This encoding involves representing each fret-string combination as a categorical variable. Given that my model predicts six fret-string positions (one for each string), the labels for each frame are encoded as a multi-dimensional array where each dimension corresponds to a string and contains a categorical encoding of the fret position.

E. Utilisation in the model

During training, the model uses these preprocessed labels to learn the mapping between CQT frames and fret-string positions. The model outputs a set of predictions for each frame, which are then compared to the preprocessed labels using a loss function designed to penalise incorrect predictions. The model's archi-

tecture (which I will describe shortly) includes a final activation layer that outputs probabilities for each fret-string position category. During inference, the model's predictions can be decoded back into fret-string combinations, allowing for the reconstruction of the played positions throughout a piece of music.

X. Network Architecture

My model, PolyTab, is an attempt at harnessing the capabilities of a Convolutional Neural Network (CNN) for the purpose of guitar tablature estimation from audio signals. The architecture is deeply inspired by notable works in the field of music information retrieval and deep-learning, specifically drawing from the foundational concepts discussed in "Guitar Tablature Estimation with a Convolutional Neural Network" by Andrew Wiggins and Youngmoo Kim, which proposes the TabCNN architecture [5].

A. Input Layer

The input to my model is a Constant-Q Transform (CQT) spectrogram of audio signals, chosen for its musical pitch linearity, allowing pitch-invariant features learning. My decision to utilise CQTs, with 192 bins spanning 8 octaves and a frame rate of approximately 43 frames per second, mirrors the data preprocessing steps in the mentioned studies, enabling effective capture of musical content with reduced dimensionality.

B. Convolutional Layers

Following the input, my model employs 3 convolutional layers with ReLU (Rectified Linear Unit) activations, which is a common choice for learning hierarchical spatial features from image data [7]. This will allow for non-linear mappings to be learned, and the ReLU activation function has been shown to train faster than alternatives [8] [9]. This design choice is motivated by the success of CNNs in various computer vision tasks [10] [11] and music information retrieval [5], where they have shown promise in tasks like musical tempo estimation, key classification, and instrument classification. The convolutional layers in my model are expected to learn features relevant to guitar tablature estimation, such as timbre differences and playability constraints inherent in the audio data. Each convolutional layer has a filter size of 3 x 3. The first layer has 32 filters and the other two each have 64. These layers are responsible for extracting features from the input spectrogram, where each filter can learn to identify specific patterns in the data. These parameters were inspired by the TabCNN model architecture.

C. Max Pooling and Dropout

A 2 x 2 max pooling layer follows the convolutional layers to introduce translation invariance and reduce the dimensionality further, a standard practice in CNN architectures [12]. This layer reduces the spatial dimensions (height and width) of the input volume for the next layers, which helps to reduce the number of parameters and computation in the network, and also helps to control overfitting by providing an abstracted form of the representation. A dropout layer with a rate of 0.25 is incorporated following the max pooling to prevent overfitting, ensuring the model's generalisability to unseen data.

D. Flattening Layer

After dropout, a flattening layer is used to transform the 2D matrix data to a vector so it can be fed into the fully connected layers.

E. Fully Connected (Dense) Layers

After flattening, the model has a dense layer with 128 units and ReLU activation. This is followed by another dropout layer with a rate of 0.5 for further regularisation. The final dense layer has num_classes x num_strings units, where num_classes is 21 (representing the number of frets, including the open string and a 'not played' class) and num_strings is 6 (representing each of the guitar strings). This implies the final dense layer has 126 units (21 x 6).

F. Reshaping and Activation

The output of the last dense layer is reshaped to 6 x 21 (num_strings x num_classes) to represent the predictions for each string and fret. At the end, a softmax activation is applied to each of the 6 strings which converts values into probabilities that sum up to 1. As such, the model outputs an estimation of six probability mass functions to represent the probability of each fret class for each string.

XI. Initial Training Strategy

Due to the output being a representation of a guitar fretboard, the problem can be viewed as 6 simultaneous multiclass classification problems. A common approach for a multiclass classification is to optimise the categorical cross-entropy between the predictions and the targets [13]. Consistent with the TabCNN approach, I computed the categorical cross-entropy for each string and summed those values. The loss function is given as

$$L(y, \hat{y}) = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{ic}\log(\hat{y}_{ic})$$

Where

- $L(y, \hat{y})$ is the loss function comparing the true labels y with the predicted labels $\hat{y}$.
- N is the number of samples in the dataset.
- C is the number of classes (in the context of guitar tabs, this could be the number of possible fret positions for each string).
- y_{ic} is the binary indicator (0 or 1) if class label c is the correct classification for observation i.
- $\hat{y}_{ic}$ is the predicted probability observation i is of class c.

This loss function sums over all classes for each observation, multiplying the true label by the log of the predicted probability, and then sums over all observations. The negative sign is used because the logarithm of values between 0 and 1 is negative, and I want the loss to be a positive quantity.

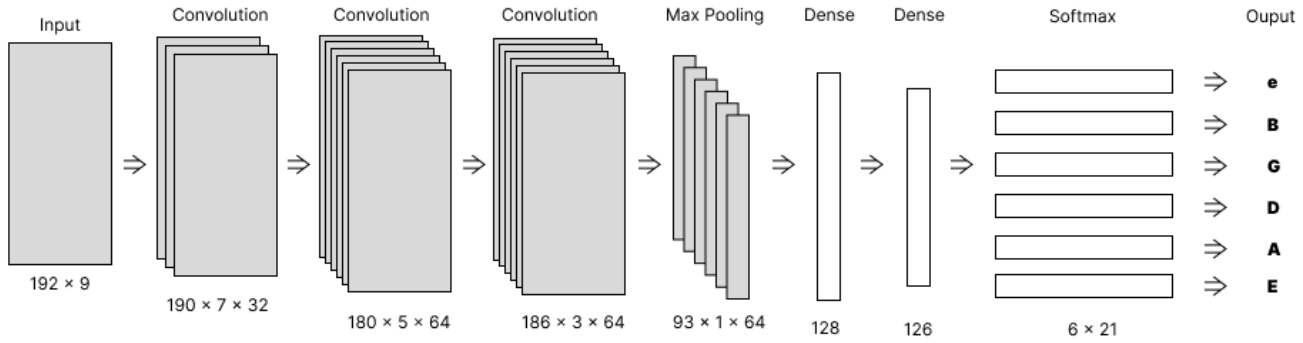


Figure 3: The architecture of the PolyTab model, illustrating the flow from the input CQT spectrogram through successive convolutional layers with ReLU activations, max pooling, dense layers, and finally to the softmax layer for multi-class classification of guitar string and fret positions.

My training procedure employs the ADADELTA optimisation algorithm, known for its robustness and adaptability, particularly in scenarios where the optimal learning rate is uncertain. This choice of optimiser, inspired by the TabCNN approach, is particularly suited for tasks like fret-string position classification in guitar performance, where the model must learn from complex, high-dimensional data.

Initial Learning Rate: ADADELTA is designed to adapt the learning rate based on the gradients' accumulated history, rendering the need for an initial learning rate less critical than in other optimisation methods. However, for completeness and adherence to best practices, specifying an initial rate (which I set to 1.0) ensures consistency and reproducibility of training sessions.

Batch Size: I've opted for a batch size of 128 samples per mini-batch, consistent with the TabCNN approach. This size is a balance between computational efficiency and the model's ability to generalise from the training data. Larger batch sizes facilitate faster computation by taking advantage of parallel processing, but may lead to a less stable convergence as the gradient estimate is based on more data.

Epochs: The model is trained over 8 epochs. An epoch is one complete pass through the entire training dataset. This number of epochs is chosen to allow sufficient model learning and adaptation to the training data without excessive computational cost or risk of overfitting.

Dropout Layers: To combat overfitting, dropout layers with a rate of 0.25 after convolutional layers and 0.5 after the dense layer (before the output layer) are employed. Dropout is a powerful regularisation technique that "drops out" a random set of activations in the specified layer by setting them to zero during training [14]. This forces the network to learn more robust features that are useful in conjunction with various different random subsets of the other neurons, enhancing the model's generalisation capability.

To further enhance the model's generalisation capability and validate its performance across different subsets of the data, I partitioned your dataset into training and validation sets based on guitarist IDs, effectively creating a form of "leave-one-guitarist-out" cross-validation.

Each fold in this setup represents a scenario where one guitarist's data is held out as the validation set while the rest contribute to the training set. This method ensures that the model's performance is evaluated in a manner that mirrors real-world application scenarios where the input data may not exactly match the training examples. The specific partitioning into folds, based on guitarist IDs, was due to the potential for overfitting to particular playing styles or recording conditions associated with individual guitarists. By ensuring that the model can generalise across different guitarists, I attempted to enhance its applicability to a broader range of musical inputs. Training across these folds involves iteratively training the model from scratch for each fold, ensuring that each validation set is unseen during its respective training phase. This rigorous process serves to underline the robustness of the resulting model.

XII. Weighted Categorical Cross Entropy Loss

Initially, I trained the model using standard categorical cross-entropy loss, adhering to the foundational approach outlined in the TabCNN paper. This phase was instrumental in establishing a baseline performance for my model. However, recognising the ordinal nature of our classification task—wherein the prediction accuracy for finger positions on the fretboard bears musical significance—I explored modifications to the loss function aimed at emphasising the musical context of misclassifications. This method is intuitively appealing because it aligns more closely with how we perceive pitch errors in music. A larger pitch error (e.g., mistaking a C for a G) is generally more disruptive than a smaller one (mistaking a C for a C#), especially in the context of guitar tablature where such mistakes can lead

to impractical or impossible fingerings.

As such, I modified the loss function to penalise larger pitch errors more heavily than smaller ones, introducing two penalty weights:

- Correct Predictions (or small errors): A reduced penalty weight of 0.5, encouraging the model to at least make closely related predictions if it cannot accurately identify the exact fret.
- Larger Errors: An initial penalty weight of 2.0, significantly penalising the model for predictions that deviate greatly from the true class.

This initial experimentation with penalty weights led to a slight deterioration in model performance compared to using the standard categorical cross-entropy loss. Recognising this, I adjusted the penalty for larger errors first to 1.5 and then to 1.2, in an attempt to find a more balanced weighting that could potentially improve the model's predictive accuracy. This experimentation with the values led to a slight improvement in model performance compared to the initial values but overall the model still performed worse than the non-weighted categorical cross entropy loss.

While this was a good first step in moving towards a more principled approach from an ad-hoc one, this experimentation did not yield the improvement I was hoping for. While weighted loss functions aim to address specific nuances in the training data, there is a risk that they might inadvertently encourage overfitting by focusing too narrowly on certain aspects of the data distribution. Regularisation techniques like dropout are crucial in mitigating this risk, but the balance between specificity and generalisation is delicate. Furthermore, the specific values chosen for the penalties (0.5, 2.0, 1.5, and 1.2) might not have been optimal for guiding the model towards better generalisation. Finding the right balance for weighted penalties often requires extensive experimentation and might benefit from a more dynamic approach rather than fixed values.

XIII. Linear Weighted Categorical Cross Entropy Loss

Leading from this idea, I attempted to implement a linear weighted categorical cross entropy loss function. Implementing a linear relationship between class errors and penalisation in the loss function was a creative approach to try and refine the model's ability to distinguish fret-string positions with a nuanced understanding of musical intervals. The idea was to dynamically adjust the penalty based on the distance between the predicted class and the true class, rather than using fixed penalty values. I attempted to map the error magnitude directly to the penalty weight, establishing a linear relationship. In this scheme, the further away the prediction was from the true class, the heavier the penalty applied, proportional to the distance.

However, during training, I observed the model hanging and training routinely stalled at the start of the fourth fold. There are several potential reasons this could have occurred:

- Computational Complexity: Dynamically calculating the penalty for each prediction could significantly increase the computational load. The function wasn't particularly optimised with any efficiency considerations, so it could slow down the training process to the point of appearing to hang, especially with large datasets or complex models. While this is likely not what was occurring, it is a possibility to consider.
- Exploding Gradient Problem: The introduction of a linearly scaling penalty could lead to issues with gradient calculation and propagation. If the penalty increases sharply with the error, it could cause gradients to become excessively large or volatile, making it difficult for the optimiser to converge to a stable solution. This is known as the exploding gradient problem [15] and it occurs when large error gradients accumulate during backpropagation, causing significant updates to the network weights. When these updates are too large, they can lead to numerical overflow, instability in the network's learning process, and ultimately prevent the network from converging to a stable solution. The exploding gradient problem is particularly prominent in deep networks or in scenarios where a penalty linearly escalates with the prediction error, leading to disproportionately large updates to the weights [16].
- Loss Surface Complexity: By introducing a direct relationship between error magnitude and penalty, the loss surface (the multi-dimensional surface representing the loss for different parameter values) could become more complex and harder to navigate. Optimisers like ADADELTA are designed to adjust learning rates based on the training process's history, but a highly irregular loss surface could challenge these mechanisms, leading to slow or stalled progress [17].

While the intention behind using a linear relationship between class errors and penalisation was to create a loss function more aligned with the musical significance of prediction errors, the practical challenges prevented the model from training and being useful in any sense. However, I was aware that I was on the right track and I was determined to design a loss function that more closely captured the musical significance of the prediction errors.

XIV. Learnable Weighted Loss

Knowing that I wanted to penalise prediction errors based on their magnitude, I decided to make the penalty weights a learnable parameter for the model.

My implementation defined a custom Keras Layer, LearnableWeightedLoss, which introduced a learnable weight parameter to adjust the penalty dynamically during the training process. This weight aims to modify the standard categorical cross-entropy loss based on the absolute difference between

the true and predicted classes. The loss weight is initialised to 1, meaning there's no initial penalty beyond the standard cross-entropy loss. This choice allows the model to start learning without any bias towards penalisation adjustments. The core idea was to adjust the cross-entropy loss dynamically by considering the absolute difference between the true and predicted classes. This approach attempted to penalise larger errors more severely than smaller ones. The weight is learnable, meaning the model itself determines the optimal way to adjust penalties during the training process through backpropagation. This adaptability can potentially lead to a more nuanced understanding and correction of errors.

The learnable aspect of the weight allowed my model to adapt its penalty mechanism based on the actual distribution and characteristics of the errors encountered during training. This can lead to a more customised loss adjustment strategy that might not be achievable through static penalty weights or with an assumed relationship between the class difference and the penalty weight. By penalising larger errors more, the model can focus its learning efforts on correcting the most significant mistakes first, potentially leading to faster convergence on challenging examples.

Indeed, this approach led to a small but significant improvement in model performance across some metrics (which I will discuss shortly) and my implementation of a Learnable Weighted Loss introduced a promising avenue for enhancing model performance on tasks with hierarchical or ordered error significance.

XV. Optimiser Experimentation

To further improve my model, I decided to experiment with using a different optimisation algorithm. While ADADELTA is a robust choice for various problems, it has limitations, particularly in handling sparse gradients and adjusting the learning rate dynamically in response to the training process [17], which can lead to slower convergence in complex tasks like music transcription. One option for this is the Adam optimiser. Adam is widely appreciated for its adaptive learning rate properties, combining the best of AdaGrad and RMSProp [18]. Adam adjusts the learning rate based on a moving average of the gradients and their square, which helps in navigating through the parameter space more efficiently. However, Adam has been criticised for not incorporating weight decay directly into the optimisation process, potentially leading to suboptimal generalisation performance. Furthermore, the lacklustre results from Seth Roberts' "Tablature Estimation of Acoustic Guitar Recordings with a Convolutional Neural Network" [19] led me away from using Adam and I decided to try the AdamW optimisation algorithm instead.

AdamW introduces a decoupled weight decay regularisation to the Adam optimiser. This separation addresses the issue with Adam's approach to L2 regularisation, which conflates weight decay with the optimiser's adaptive learning rate mechanism [20]. By decoupling the weight decay, AdamW directly regularises the weights, leading to better training dynamics and generalisation performance. The key advantages of using AdamW over ADADELTA (and even Adam) for my model include:

- Enhanced Regularisation: AdamW's decoupled weight decay promotes a more straightforward regularisation of model parameters, potentially leading to improved generalisation by mitigating overfitting.
- Improved Convergence: By correcting the integration of weight decay, AdamW may facilitate a more efficient convergence towards the optimal set of parameters, especially in complex models such as those involved in automatic music transcription.
- Flexibility and Control: AdamW provides more control over the regularisation process, allowing for fine-tuning of the model in a way that aligns with the unique challenges of guitar tablature transcription.

The switch to AdamW in theory will yield several improvements in my model:

- Better Generalisation: The direct regularisation might help my model generalise better to unseen data, a crucial aspect of transcription models that deal with diverse musical styles and guitar techniques.
- Faster Convergence: With an improved optimisation path, your model may converge faster, reducing training time without compromising the accuracy of the fret-string position classification.
- Robustness to Overfitting: The decoupled weight decay mechanism of AdamW can offer a more robust defence against overfitting, essential when working with limited or highly specialised datasets like GuitarSet.

By leveraging AdamW's advantages, I aimed to enhance my CNN model's performance, efficiency, and generalisation capabilities, and indeed, using AdamW led to significant improvements in some of my metrics.

XVI. Metrics

To evaluate the performance of my model designed for guitar fret-string position classification I employed several metrics from the TabCNN paper that extend from traditional multipitch estimation metrics [21]. These metrics are tailored to assess the accuracy and reliability of the predicted tablature against ground truth data. Each metric provides unique insights into the model's performance, highlighting areas of strength and opportunities for improvement.

A. Pitch Precision

Pitch precision measures the accuracy of pitch class predictions, focusing on the model's ability to correctly identify pitch classes present in the music piece. This metric is crucial for

assessing the model's effectiveness in recognising pitches, irrespective of their specific string or fret positioning.

$$\text{Pitch Precision} = \frac{\sum \text{True Positive Pitches}}{\sum \text{Predicted Pitches}}$$

B. Pitch Recall

Pitch recall evaluates the model's capacity to identify all relevant pitch classes within the ground truth tablature. It provides insights into the model's sensitivity towards detecting pitches, highlighting any tendencies to miss or overlook pitch classes.

$$\text{Pitch Recall} = \frac{\sum \text{True Positive Pitches}}{\sum \text{Ground Truth Pitches}}$$

C. Pitch F-Measure

The pitch F-measure combines precision and recall into a single metric, offering a balanced view of the model's performance in terms of both accuracy and completeness in pitch class prediction.

$$\text{Pitch F-Measure} = 2 \times \frac{\text{Pitch Precision} \times \text{Pitch Recall}}{\text{Pitch Precision} + \text{Pitch Recall}}$$

D. Tab Precision

Tab precision assesses the accuracy of the binary tablature predictions, focusing on the model's ability to correctly identify fretted positions across all strings. This metric is essential for understanding the model's precision in predicting finger placements on the guitar fretboard.

$$\text{Tab Precision} = \frac{\sum \text{True Positive Frets}}{\sum \text{Predicted Frets}}$$

E. Tab Recall

Tab recall measures the model's capability to capture all fretted positions as per the ground truth tablature. It evaluates the model's sensitivity in identifying correct finger placements, emphasising the avoidance of missed or ignored frets.

$$\text{Tab Recall} = \frac{\sum \text{True Positive Frets}}{\sum \text{Ground Truth Frets}}$$

F. Tab F-Measure

Similar to the pitch F-measure, the tab F-measure offers a harmonised metric that encapsulates both the precision and recall of binary tablature predictions. It serves as a comprehensive gauge of the model's overall performance in predicting accurate finger placements.

$$\text{Tab F-Measure} = 2 \times \frac{\text{Tab Precision} \times \text{Tab Recall}}{\text{Tab Precision} + \text{Tab Recall}}$$

G. Tab Disambiguation Ratio

The tab disambiguation ratio compares the precision of binary tablature predictions with the precision of pitch class predictions. It offers insights into the model's ability to disambiguate between multiple potential finger placements that produce the same pitch.

$$\text{Tab Disambiguation Ratio} = \frac{\text{Tab Precision}}{\text{Pitch Precision}}$$

These metrics collectively offer a multifaceted evaluation of the model's performance, providing a thorough understanding of its capabilities and limitations in the context of guitar tablature prediction.

XVII. Results

PP	PR	PF	TP	TR	TF	TDR
0.862	0.197	0.316	0.636	0.153	0.243	0.737

Table 1: Summary of results for the Non Weighted implementation.

PP	PR	PF	TP	TR	TF	TDR
0.822	0.180	0.289	0.607	0.142	0.224	0.739

Table 2: Summary of results for the Hardcode Weighted 2.0 implementation.

PP	PR	PF	TP	TR	TF	TDR
0.797	0.154	0.255	0.598	0.121	0.199	0.751

Table 3: Summary of results for the Hardcode Weighted 1.5 implementation.

PP	PR	PF	TP	TR	TF	TDR
0.879	0.229	0.358	0.616	0.174	0.267	0.700

Table 4: Summary of results for the Learnable Weighted Loss implementation.

PP	PR	PF	TP	TR	TF	TDR
0.846	0.605	0.701	0.685	0.506	0.578	0.810

Table 5: Summary of results for the Learnable Weighted Loss with AdamW implementation.

From the results presented in the tables, we can dissect the impact of each methodical tweak on my model's performance metrics: pitch precision (PP), pitch recall (PR), pitch F-measure (PF), tab precision (TP), tab recall (TR), tab F-measure (TF), and tab disambiguation ratio (TDR).

Starting with the Non Weighted foundational approach (Table 1), we notice baseline results with modest recall values, which indicates a conservative prediction model less likely to predict notes that weren't there (false positives), yet also likely missing out on some true notes (false negatives). Precision values are higher, which means when the model predicts a note, it does so with reasonable confidence.

Moving to Hardcode Weighted Loss strategies (Tables 2 and 3), where specific penalties were applied to the loss function in an attempt to adjust the model's prediction sensitivity, this approach didn't lead to significant improvements. The "Hardcode Weighted 2.0" implementation marginally improved precision but at the cost of recall. When the weight was adjusted to 1.5, the precision dropped below the non-weighted approach,

suggesting that manual weighting did not align well with the complex dynamics of tablature prediction. This is indicative of a model struggling to generalise; overly penalised mistakes discouraged the model from making bolder predictions that could have been correct.

The introduction of the Learnable Weighted Loss (Table 4) marked a turning point in my experimentation. The learnable aspect allowed the model to adjust its penalty dynamically during training, leading to a balanced approach to predicting notes. It's notable that the recall and F-measure values for both pitch and tab saw a significant increase, suggesting the model became better at not missing true positives while maintaining a solid precision. This approach is more in tune with the intricacies of guitar tablature prediction, as the model can learn the optimal trade-off between precision and recall during training.

The leap in performance is most striking with the addition of the AdamW optimiser (Table 5). AdamW is known for handling weight decay in a way that's more suitable for adaptive learning-rate algorithms, which might have been instrumental in the significant gains in all metrics. Notably, the recall values saw a dramatic increase, implying that the model became much more adept at detecting notes that are present. This did not come at the expense of precision, which also saw improvements. The result is a model that confidently predicts a greater number of correct notes without a proportional rise in false positives.

Incorporating AdamW with the learnable weighted loss seems to have given the model the ability to better navigate the balance between confidently predicting notes and minimising false detections. This could be a result of AdamW's sophisticated handling of sparsity in data and its effectiveness in complex scenarios where the relationship between inputs and outputs is not straightforward.

My experimentation and iterative refinement clearly demonstrate an improvement over the foundational TabCNN approach, as reflected in the final values. The model's methodology appears to have evolved to a point where it not only learns from the data but also adapts its learning process, culminating in a model that outperforms its predecessors in both pitch and tab prediction tasks, ultimately providing a more accurate and reliable tool for guitar tablature prediction.

XVIII. Temporal Smoothing and Tablature Output

I have also included a PolyTabPredictor class that capitalises on the convolutional neural network's ability to learn from spectral representations of audio signals to transcribe audio into guitar tabs.

The methodology for this is the same as the model itself. It starts with preprocessing the audio file, which involves normalisation and downsampling for dimensionality reduction. I then take the processed audio file and generate a CQT representation for it to pass into the model.

The predictor then uses this CQT representation to predict tablature in real-time frames, utilising the convolutional neural network. This part of the process tackles the task of directly mapping the audio signal to guitar tablature.

The PolyTabPredictor then aggregates predictions over a window to smooth out the tablature prediction, a concept suggested in the TabCNN paper regarding temporal smoothing and frame aggregation as further work. This step is key as it reduces the noise in predictions, resulting in a cleaner, more accurate transcription. The approach I've taken not only identifies the individual notes played but also provides context through the aggregation window, giving a more nuanced and temporally coherent transcription through the following steps.

Frame-by-Frame Analysis: The model predicts the likelihood of fretting for each string at each moment in time. This produces a sequence of predictions (frames) that individually represent the most likely tablature for each small segment of the audio.

Aggregation Window: I introduce the concept of an 'aggregation window', which spans several consecutive frames. This window is a crucial component of the temporal smoothing process, as it considers not just the immediate frame but also the context provided by surrounding frames.

Majority Voting within the Window: For each string within the aggregation window, I look at the series of predictions and apply a 'majority vote' approach. This means that for each string, the fretting that occurs most frequently within that window is selected as the final prediction for that segment.

Continuous Tablature Stream: By sliding this window across the entire sequence of frame predictions and applying the majority vote rule, I create a smoothed-out, continuous stream of tablature that reduces the granularity of instantaneous predictions. This method effectively filters out spurious predictions that could be the result of noise or transient features in the audio signal.

Resulting Aggregated Tabs: The aggregated predictions now represent a more temporally consistent and musically coherent transcription, likely to align more closely with how a guitarist would actually play the piece.

The predictor then goes beyond the foundational work by not only generating tablature for individual frames but also by creating a text-based and visual representation of the tablature. The generation of images of guitar tabs for visualising the tablature is particularly noteworthy as it provides a user-friendly way to understand and utilise the model's predictions, which aligns with the goal of making the transcription process accessible.

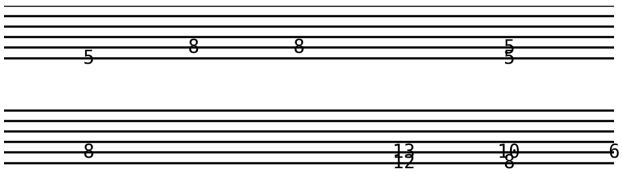


Figure 4: Excerpt from the generated tablature for the file "05_SS3-98-C_solo_mic.wav"

By implementing this temporal smoothing technique, my PolyTabPredictor mitigates the issue of frame-to-frame volatility, which is common in frame-based prediction models [22]. This approach not only improves the accuracy of the transcription but also results in a more playable tablature, considering the physical constraints and playability on the guitar.

XIX. Critical Appraisal

My project aimed to develop a machine-learning model capable of transcribing isolated polyphonic guitar recordings into tablature with a degree of accuracy. Utilising Convolutional Neural Networks (CNNs), my approach aligns with contemporary strides in Music Information Retrieval (MIR), particularly those leveraging deep-learning methodologies. The results indicated by metrics such as pitch precision, pitch recall, tab precision, and tab recall suggest that my model, PolyTab, has made tangible advancements in the domain of automatic music transcription.

Critically comparing PolyTab to its predecessors and contemporaries, such as TabCNN, it's evident that my project has not only drawn inspiration from these prior works but also has sought to overcome their limitations through innovative techniques. For instance, my introduction of a learnable weighted loss function and subsequent experiments with optimisers like AdamW reflects an iterative, nuanced approach to problem-solving that is characteristic of rigorous scientific inquiry.

However, the robustness of these improvements can only be thoroughly assessed in broader application contexts. The model's generalisability across various musical genres, playing styles, and recording qualities remains an empirical question. Furthermore, while the reported improvements in precision and recall are encouraging, these are initial findings and must be tested in real-world scenarios where audio quality and musician idiosyncrasies introduce a high degree of variability.

My exploration into the model's training strategies, particularly the move from a static weighted categorical cross-entropy loss to a learnable and more dynamic version, showcases a deep understanding of the challenges inherent in MIR tasks. Nevertheless, it is equally important to acknowledge the limitations and difficulties faced during development, such as the initial worsening of model performance with weighted loss functions and computational issues encountered during the implementation of the linear weighted categorical cross-entropy loss function.

In terms of innovation, the temporal smoothing technique and the PolyTabPredictor class demonstrate a commitment to enhancing the practicality and usability of the model's outputs. The creative step of visualising the tablature output not only aids in interpretability but also enhances the potential for practical application by musicians.

When positioned against related work in the public domain, my project contributes meaningfully to the field. However, it would benefit from further validation against a wider array of datasets and in comparison with other state-of-the-art models.

XX. Conclusion

This project embarked on an ambitious journey to bridge the gap between polyphonic audio signals and their symbolic transcription into guitar tablature. By leveraging Convolutional Neural Networks, I sought to automate the process of music transcription, a task traditionally characterised by its time-consuming and labor-intensive nature.

Throughout the project, I developed PolyTab, a model designed to transcribe polyphonic guitar recordings with noteworthy accuracy. The model's architecture was influenced by leading-edge research in the field, yet it introduced novel elements such as learnable weighted loss and an aggregation window for temporal smoothing. These innovations were aimed at enhancing the model's predictive capabilities and ensuring the output's fidelity to the characteristics of human guitar playing.

Key findings from this project include a demonstrable improvement in pitch and tab precision and recall, outperforming the established TabCNN model. My results indicated that the model could successfully translate audio to tablature, maintaining a high degree of accuracy in its predictions. The inclusion of a PolyTabPredictor class further extended the project's contributions, providing a tool for real-time tablature prediction that is accessible to musicians and researchers alike.

Despite these achievements, I acknowledge the constraints of the dataset and the limitations posed by computational challenges during the model's development. Future work will look to expand the model's dataset diversity, test its generalisability across different musical styles and recording conditions, and compare its performance with other state-of-the-art models.

In conclusion, my project stands as a significant step towards more accessible and automated guitar transcription, with promising results that warrant further investigation and application. It is imperative to continue refining the model, rigorously testing its predictions, and expanding its robustness to ensure that it remains not only academically relevant but also genuinely useful to the guitar-playing community.

APPENDIX A

UNIVERSITY OF ST ANDREWS
TEACHING AND RESEARCH ETHICS COMMITTEE (UTREC)
SCHOOL OF COMPUTER SCIENCE
PRELIMINARY ETHICS SELF-ASSESSMENT FORM

This Preliminary Ethics Self-Assessment Form is to be conducted by the researcher, and completed in conjunction with the Guidelines for Ethical Research Practice. All staff and students of the School of Computer Science must complete it prior to commencing research.

This Form will act as a formal record of your ethical considerations.
Tick one box

- ☐ **Staff Project**
- ☐ **Postgraduate Project**
- ☒ **Undergraduate Project**

Title of project

Automatic Polyphonic Guitar Transcription

Name of researcher(s)

Vanaj Krishna Moorthy

Name of supervisor (for student research)

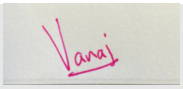
Ognjen Arandelovic

OVERALL ASSESSMENT (to be signed after questions, overleaf, have been completed)

Self audit has been conducted **YES** ☒ **NO** ☐

There are no ethical issues raised by this project

Signature Student or Researcher



Print Name

VANAJ KRISHNA MOORTHY

Date

25/09/2023

Signature Lead Researcher or Supervisor

Print Name

Ognjen Arandjelovic

Date

25/09/2023

Appendix A

This form must be date stamped and held in the files of the Lead Researcher or Supervisor. If fieldwork is required, a copy must also be lodged with appropriate Risk Assessment forms. The School Ethics Committee will be responsible for monitoring assessments.

APPENDIX A

Computer Science Preliminary Ethics Self-Assessment Form

Research with secondary datasets

Please check UTREC guidance on secondary datasets (https://www.st-andrews.ac.uk/research/integrity-ethics/humans/ethical-guidance/secondary-data/ and https://www.st-andrews.ac.uk/research/integrity-ethics/humans/ethical-guidance/confidentiality-data-protection/). Based on the guidance, does your project need ethics approval?

YES ☐ **NO** ☒

** If your research involves secondary datasets, please list them with links in DOER.*

Research with human subjects

Does your research involve collecting personal data on human subjects?

YES ☐ **NO** ☒

If YES, full ethics review required

Does your research involve human subjects or have potential adverse consequences for human welfare and wellbeing?

YES ☐ **NO** ☒

If YES, full ethics review required
For example:
Will you be surveying, observing or interviewing human subjects?
Does your research have the potential to have a significant negative effect on people in the study area?

Potential physical or psychological harm, discomfort or stress

Are there any foreseeable risks to the researcher, or to any participants in this research?

YES ☐ **NO** ☒

If YES, full ethics review required
For example:
Is there any potential that there could be physical harm for anyone involved in the research?
Is there any potential for psychological harm, discomfort or stress for anyone involved in the research?

Conflicts of interest

Do any conflicts of interest arise?

YES ☐ **NO** ☒

If YES, full ethics review required
For example:
Might research objectivity be compromised by sponsorship?
Might any issues of intellectual property or roles in research be raised?

Funding

Is your research funded externally?

YES ☐ **NO** ☒

If YES, does the funder appear on the 'currently automatically approved' list on the UTREC website?

YES ☐ **NO** ☐

APPENDIX A

If NO, you will need to submit a Funding Approval Application as per instructions on the UTREC website.

Research with animals

Does your research involve the use of living animals?

YES ☐ **NO** ☒

If YES, your proposal must be referred to the University's Animal Welfare and Ethics Committee (AWEC)

University Teaching and Research Ethics Committee (UTREC) pages
http://www.st-andrews.ac.uk/utrec/

User Manual

To run the project and train the model yourself a few step must be taken.

- I have not submitted the GuitarSet folder due to it's large size but the dataset can be downloaded here. Make sure to unzip this folder and place it in the root directory of the project.
- After that, you must activate a virtual environment to install the dependencies. This can be done with

  ```
  python3 -m venv <env>
  ```

  . Then you can activate the virtual environment by running

  ```
  source <env>/bin/activate
  ```

- Next, please install the dependencies with

  ```
  pip install -r requirements.txt
  ```

- After this, you have to generate the CQT representations for the audio files using

  ```
  python3 ParallelGenerateCQTs.py
  ```

- And finally you can train the model with

  ```
  python3 PolyTab.py
  ```

- Once the model has trained, you can run

  ```
  python3 PolyTabPredictor.py --weights "path/to/weights.h5"
     --audio "path/to/audio/file.wav"
  ```

  with the trained weights and the audio you want to predict for. The saved predictions can be found in the /predictions folder.

- I have submitted my trained model weights as well, the most relevant of which are in "saved/c 2024-03-21 171741/5/weights.h5". Therefore, you can run

  ```
  python3 PolyTabPredictor.py --weights
     "saved/c 2024-03-21 171741/5/weights.h5" --audio "path/to/audio/file.wav"
  ```

  to predict using the model which was trained with the learnable weighted loss and AdamW optimiser.

REFERENCES

[1] A. L. Opdahl, *Computer-aided music transcription*, vol. 2. Springer Science & Business Media, 2011.

[2] Q. Xi, R. Bittner, J. Pauwels, X. Ye, and J. P. Bello, "Guitarset: A dataset for guitar transcription," in *19th International Society for Music Information Retrieval Conference*, pp. 313–317, 2018.

[3] C. E. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949.

[4] X. Serra, *Music and machine learning*. Springer Science & Business Media, 2012.

[5] A. Wiggins and Y. Kim, "Guitar tablature estimation with a convolutional neural network," in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, Drexel University, Dept. of Electrical and Computer Engineering, 2019.

[6] Y. Bengio, "Practical recommendations for gradient-based learning of deep architectures," in *Neural networks: Tricks of the trade*, vol. 1, pp. 432–448, Springer, 2012.

[7] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[8] V. Nair and G. E. Hinton, "Rectified linear units improve unsupervised pre-training for deep neural networks," in *Proceedings of the 27th international conference on neural information processing systems (NIPS'10)*, vol. 1, pp. 807–814, MIT Press, 2010.

[9] B. Xu, N. Wang, T. Tian, J. Yong, Y. Gao, and D. Xu, "Empirical evaluation of rectified activations in convolutional networks," May 2015.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural networks*, vol. 25, no. 1, pp. 1097–1105, 2012.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," September 2014.

[12] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*. MIT press, 2015.

[13] C. M. Bishop, *Pattern recognition and machine learning*. Springer, 2006.

[14] N. Srivastava, G. E. 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. 1, pp. 1929–1958, 2014.

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[16] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," *Proceedings of the 30th International Conference on Machine Learning (ICML)*, vol. 28, no. 3, pp. 1310–1318, 2013.

[17] M. D. Zeiler, "Adadelta: An adaptive learning rate method," December 2012.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," December 2014.

[19] S. Roberts, "Tablature estimation of acoustic guitar recordings with a convolutional neural network," technical report, Hajim School of Engineering and Applied Sciences, 2022.

[20] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," November 2017.

[21] M. Hutchens, *Music Information Retrieval*. Springer, 2005.

[22] M. Müller, M. Schindler, and S. Behnke, "Multimodal deep learning for acoustic guitar transcription," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1475–1479, IEEE, 2018.