Music Tempo Estimation from EEG Data

Final Project Writeup
CSCE 470

Eleonora Stadtmüller Caballero, Jaren Ramirez, Quinton Odenthal

May 2, 2024

Table of Contents

Abstract

Although music tempo estimation is a well-established task in the Music Information Retrieval (MIR) field, it typically relies solely on audio data analysis, not taking into account the human brain's response to the music. In our project, we explore a new approach to music tempo estimation by analyzing data from the Naturalistic Music EEG Dataset - Tempo (NMED-T), which consists of electroencephalogram (EEG) data collected from participants who listened to a set of songs. We propose that deep learning can find features in the brain data that can be directly correlated to music tempo and use three separate deep learning models to showcase this. We discuss the results of each of the models and outline future work that can be done to improve their performance.

1. Introduction

Tempo estimation is a fundamental MIR task with a wide range of practical applications that, among others, include exercise tempo synchronization or smart playlist generation [1]. It benefits from the existence of a stable tempo, which is often present in Western music like pop, rock, and electronic dance music. When music features this consistent and steady beat, individuals, particularly those with musical training, often find it easy to pick up on and clap or tap along with the beat.

Similarly, a computer is able to easily estimate the tempo of a song with a steady beat by processing its audio data. A variety of lower level signal processing and higher level machine learning techniques have been developed to perform tempo estimation or beat tracking from music recordings, and are what is generally used in practice for any type of tempo estimation applications [2].

One approach that has been under-researched in the MIR field is music tempo estimation from brain wave data. While machine learning and deep neural networks have proven to be an effective way to track the tempo in audio data, these techniques have rarely been applied to brain data.
The challenges of noisy data and limited datasets may hinder the application of machine learning in this field. However, pursuing this research is justified given the established knowledge that the human brain synchronizes with rhythmic sequences, a phenomenon observable in EEG data through increased voltage magnitudes. In this research, we seek to leverage this property of the brain to build models that enable us to correctly identify the tempo of a song that a person was listening to.

2. The Data

2.1. Dataset

The NMED-T dataset [3] consists of dense-array EEG responses from 20 adult participants who each listened to 10 songs (Table 1). These songs were selected for their steady beat and were of a familiar (Western) musical tradition to the participants. Most participants had musical training and listened to music regularly. Data collection was performed by the Center for Computer Research in Music and Acoustics (CCRMA) at Stanford University in a comfortable and isolated environment where music was presented to the participants through speakers [4].

#	Song Title	Artist	Tempo (bpm)	Tempo (Hz)	min:sec	Datapoints
1	First Fires	Bonobo	55.97	0.9328	4:38	34750
2	Oino	LA Priest	69.44	1.1574	4:31	33875
3	Tiptoes	Daedelus	74.26	1.2376	4:36	34500
4	Careless Love	Croquet Club	82.42	1.3736	4:54	36750
5	Lebanese Blonde	Thievery Corporation	91.46	1.5244	4:49	36125
6	Canopée	Polo & Pan	96.15	1.6026	4:36	34500
7	Doing Yoga	Kazy Lambist	108.70	1.8116	4:52	36500
8	Until the Sun Needs to Rise	Rüfüs du Sol	120.00	2.0000	4:52	36500
9	Silent Shout	The Knife	128.21	2.1368	4:54	36750
10	The Last Thing You Should Do	David Bowie	150.00	2.5000	4:58	37250

Table 1. Table of songs and tempos in the NMED-T dataset. All ten songs range between 4:30 to 5:00 minutes and 55 to 150 beats per minute (bpm). Songs are listed in ascending order of tempo.

After data collection, the EEG responses were then cleaned and preprocessed, downsampled to a sampling rate of 125 Hz, and made available to download in Matlab format. Each song's EEG data is stored in a separate Matlab file, where 24 bit numeric values representing microvolts are arranged in an array of size 125 x T x 20 (electrodes x song time x participant). Although the raw EEG data from the NMED-T dataset is also provided, only the cleaned and preprocessed data will be used to train the machine learning models for this project, as custom cleaning of the data is beyond our scope.

2.2. Further Data Processing

The following additional preparation steps were performed on the data before it could be input into the machine learning models.

First, reliable component analysis (RCA) was performed in Matlab on each of the 10 songs' 3-dimensional data matrix in order to reduce its dimensionality along the electrode dimension. This took the data from a *125 x song time x 20* matrix to a *song time x 20* matrix for each song (Figure 1). RCA is a signal processing technique shown to reduce the signal-to-noise ratio of EEG data while also reducing the dimensionality of the data by extracting the maximally reliable component of the electrode dimension [5]. By using RCA, we hoped to speed up the training time of the model and increase performance without losing important information.

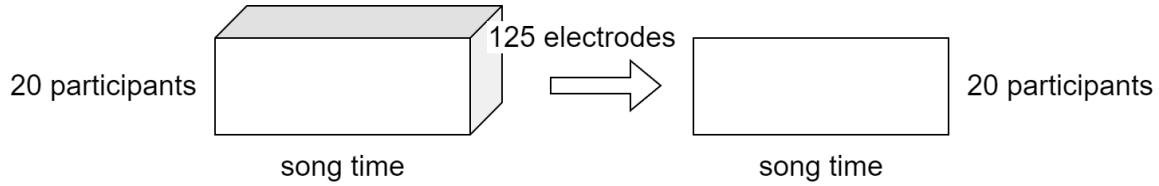


Figure 1. Dimensionality reduction of the data along the electrode dimension using RCA.

Second, a uniform amount of data was retained across each song in order for the models to be able to ingest the entire dataset without making adjustments to the model architecture each time. Per song, a uniform 4 minutes of data were retained starting from 15 seconds into the song, enough time for the participants to pick up on the beat [3]. This uniform sized data was exported from Matlab as a .mat file, with the 'data' variable storing the *song time x 20* matrix.

Following this, the data from all the songs was aggregated into csv files using a Python script. The output data frame was of size *200 x song time*. During this step, data was labeled for classification, resulting in two separate csv files, one where the 'label' column contained the labels for 3 classes and the other where it contained the labels for 5 classes (Table 2). First, the 3-class model was created to differentiate between slow, medium, and fast tempo songs. Then, the model was fine tuned to differentiate between 5 classes of tempos, which is half the number of songs in the dataset. Because the tempos of the songs were not equidistant from each other across the dataset, labeling resulted in unequal representation of songs within each class, which was amended by class weighting.

#	Song Title	Tempo (bpm)	3 classes	5 classes
1	First Fires	55.97	0	0
2	Oino	69.44	0	0
3	Tiptoes	74.26	0	1
4	Careless Love	82.42	0	1
5	Lebanese Blonde	91.46	1	2
6	Canopée	96.15	1	2
7	Doing Yoga	108.70	1	2
8	Until the Sun Needs to Rise	120.00	1	3
9	Silent Shout	128.21	2	3
10	The Last Thing You Should Do	150.00	2	4

Table 2. Distribution of class labels for the classification model across the ten songs.

Third, each sample row (vector) of data was centered around the mean and then normalized using L2 normalization, which is a technique used to scale vectors by dividing each vector component by the Euclidean norm, ensuring that the resulting vectors have a magnitude of 1 (Figure 2). By normalizing the data in this way, we hoped to reduce the influence of outliers in our dataset and make the data more interpretable by the model algorithms.

$$|\mathbf{x}| = \sqrt{\sum_{k=1}^{n} |x_k|^2},$$

Figure 2. L2 normalization, where |x| represents the Euclidean norm of each vector.

Fourth, we implemented the option to synthetically augment the size of the dataset during model training by splitting each sample into equally large sections (Figure 3). Due to the small size of our dataset (200 samples) and the low variety in songs, we hoped that increasing the size of the dataset would help counteract overfitting. This method of slicing was possible because all ten songs exhibited a steady constant beat, meaning that the global tempo of the song would not be different from local tempos within each section.

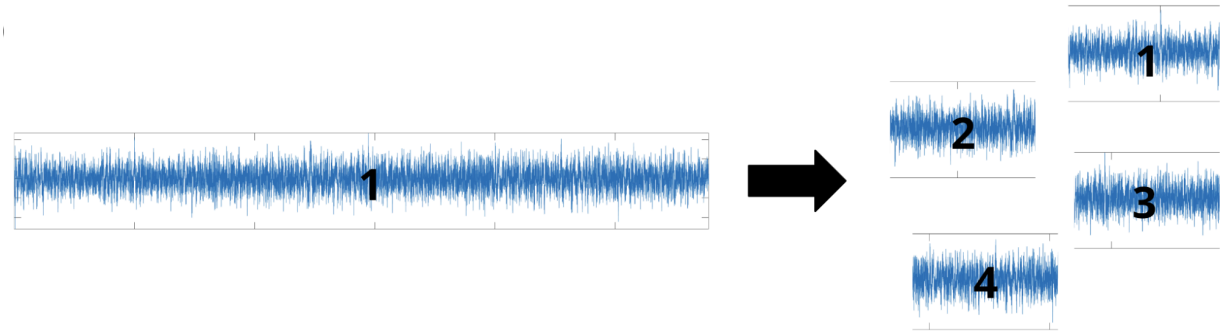


Figure 3. Example of synthetic data augmentation by slicing a sample into four equally large chunks.

Lastly, we performed k-fold cross-validation while training the models to assess their performance across different subsets of the dataset and gather results that were less dependent on a certain dataset split. Before model training, the training set was divided into 10 folds - one per song - and cross validation was performed on each fold (Figure 4). We specifically created the folds this way to also get an idea of how well the models would generalize to a new song they hadn't encountered before during training.

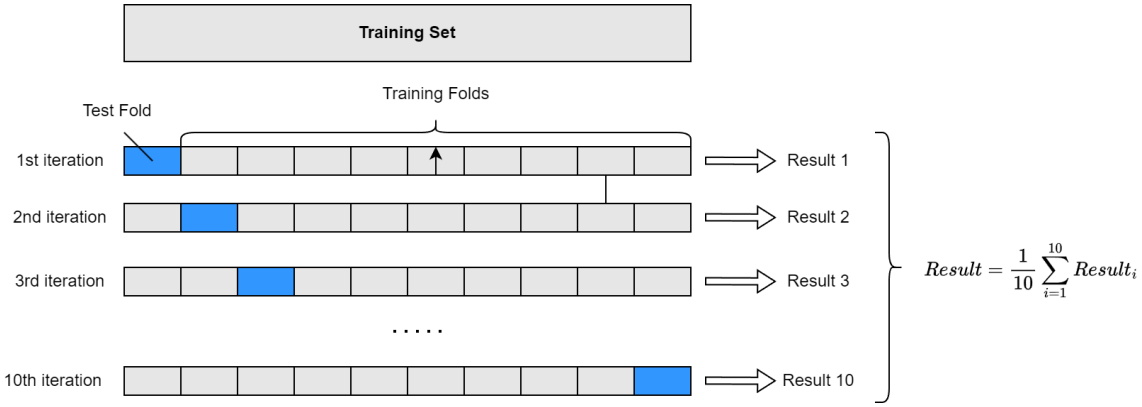


Figure 4. K-fold cross validation with 10 folds.

2.3. Testing Data Validity

To ensure that our data was being transformed properly after each step we would print out test values and ensure that they were properly transformed. Further advancements on this would likely want test cases to ensure that it was properly changed. After any data splitting we would also need to ensure that the shape of the newly formed data was what we expected. This was simple as it just required a print out of the shape and comparing the new shape with the expected (*window size x frequency*).

3. Methodology

3.1. Model Selection

Three separate deep learning models were created to perform tempo estimation on the dataset. A Multilayer Perceptron (MLP) was used for multiple linear regression. A 2-dimensional Convolutional Neural Network (CNN) was used for regression. A 1-dimensional CNN was used for multiclass classification.

Before we continue into the specifics of the models we would like to give an overview of what the specific machine learning models are and why they can be useful for this type of data. An MLP is a type of artificial neural network that is fully connected. These layers are each a complete bipartite graph $K_{m,n}$ where m and n are the sizes of the connected layers between each layer. These are useful for finding non linear relationships in a dataset. A CNN is a neural network whose architecture builds upon the MLP. A CNN adds convolutional layers on top of the MLP's fully connected layers. These convolutional layers apply a sliding dot product of customizable size to the data and are useful for finding spatial and temporal features.

3.2. MLP for multiple linear regression

The multilayer perceptron implemented multiple linear regression on the Reliable Component Analysis (RCA) filtered EEG data. It processed this data through an input layer comprising 30,000 nodes, which was fully connected to a dense layer consisting of 128 nodes. In this dense layer, a linear activation function was applied to the nodes. Then, the dense layer was fully connected to the output layer, which comprised a single node (refer to Figure 5). This model was selected to figure out how accurately a relatively non-complex model would be able to predict song tempos.

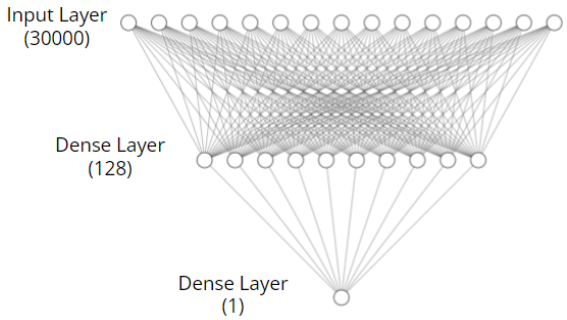


Figure 5. Architecture of the MLP for multiple linear regression

3.3. 2D CNN for regression

The 2-dimensional CNN used for regression used the data across all of the electrodes instead of using the Reliable Component of each of the song time samples. The model architecture was quick to overfit when made too complicated or dense and underfit when it was not complicated enough. The line between these model types was small and even the smallest of changes had deep impacts on the final outcomes. The model that we had the best results with consisted of two convolutional layers and two max pooling layers between both of them as shown in Figure 6. The output from the max pooling layer was flattened and fed into a dense layer of 256 neurons which was fully connected to a dense layer of 128 neurons, which was fed into the output neuron. To help mitigate the problem of overfitting I chose an aggressive dropout rate of 40% between every layer. Previous research has shown that 2D CNN's of similar complexity have been successfully used on EEG data [6].

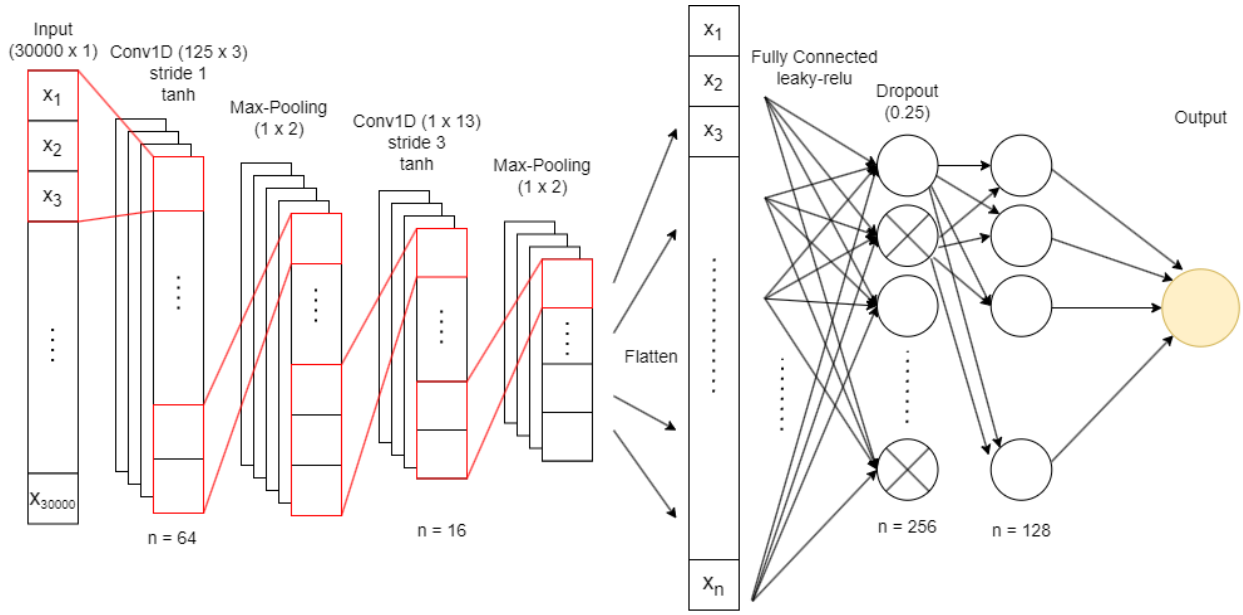


Figure 6. Model architecture of the 2-dimensional CNN for regression.

3.4. 1D CNN for multiclass classification

1-dimensional convolutional neural networks excel in processing time-series data while requiring less computational power than other methods used for the same purpose. Because time-series data is required, the 1-dimensional RCA-filtered data was used at the input layer. As mentioned before, classification was first attempted with three labels for the song tempo (slow = 0, medium = 1, fast = 2)

to establish a baseline model. The hyperparameters of the 3-class model were fine-tuned and the same model was used with 5 labels to create a more precise tempo classifier.

The architecture of the 3-class model is shown in Figure 7. It consists of four hidden layers: two convolutional layers with max pooling and two dense layers with dropout. The output of the last dropout layer led into three neurons which produced the probabilities for a sample to be in each of the three classes. The model architecture for the 5-class model is the same, except that its output layer contains five neurons.

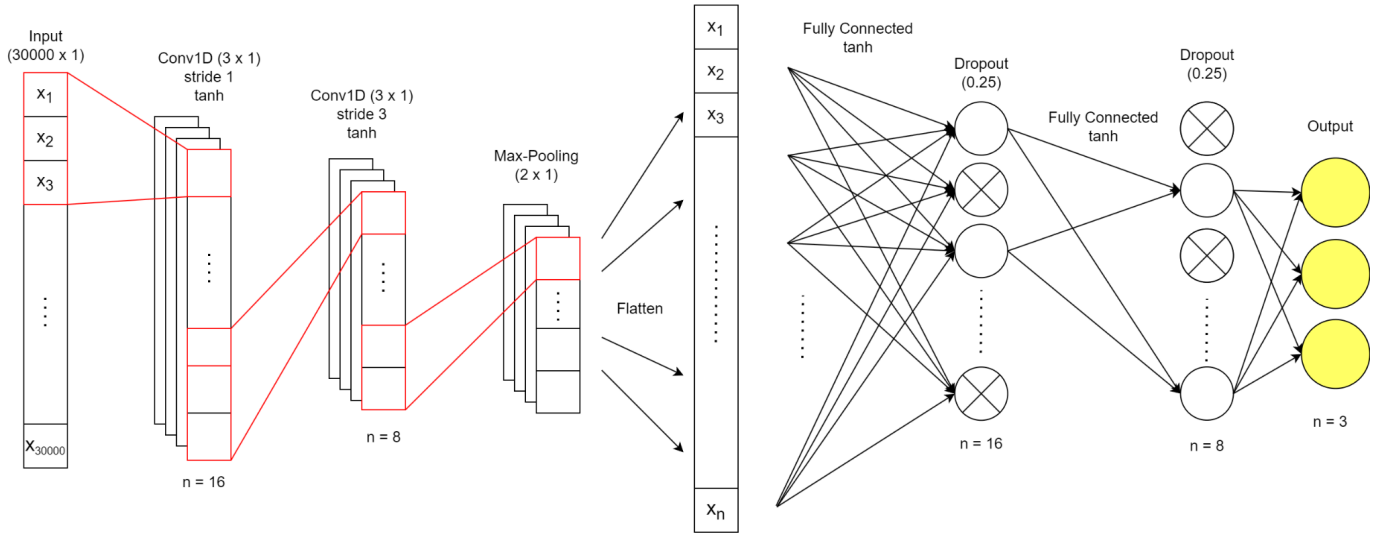


Figure 7. Model architecture of the 1-dimensional CNN for 3-class classification.

For both models, hyperparameters were tuned to the following:

- The kernel configuration in the two convolutional layers was 16 kernels of size 3 at a stride of 1 and 8 kernels of size 3 at a stride of 3.
- The pool size in the max pooling layers was set to 2.
- The number of kernels in the two dense layers was 16 and 8.
- Hyperbolic tangent (tanh) was used as the activation function for all hidden layers since we were working with a normalized dataset.
- Adam was used as the optimizer.
- The dropout rate was set to 25%.

4. Results

The results were promising for all three models. As expected, the 2D CNN had better performance for regression than the MLP, as it was able to capture more complex patterns using the convolutional layers. We found that with all three models there was a trend to predict more accurately in the median tempo range of the dataset, which is where the data density was greatest. Meanwhile, the error would increase drastically approaching the fastest tempos and would increase mildly approaching the slowest tempos. This is in line with the distribution of data across the tempo range of the dataset. The loss function used was mean squared error (MSE) for regression and sparse categorical cross-entropy for classification.

4.1. MLP for multiple linear regression results

The multilayer perceptron implementing multiple linear regression showed the worst results between the two regression models. The mean squared error on the test data was 827.1. As you can see in figure 8, the model was very inconsistent. There are many outliers and generally inaccurate predictions. However, when training time is taken into account the model performance becomes much more appealing. The model trained itself on the data in only 5.16 seconds. This is the fastest training time among the three models we created. This shows the benefit of this kind of model very well. It is simple and fast, yet potentially inaccurate.

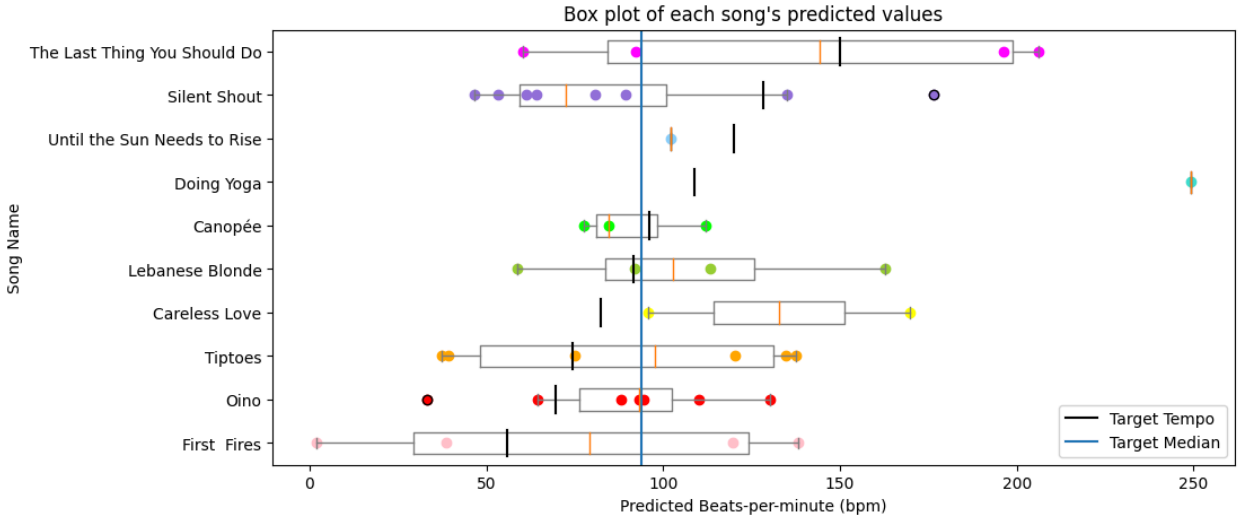


Figure 8. Dot and box plot of MLP for multiple linear regression results.

4.2. 2D CNN for regression results

The results for the 2D CNN did well in the high density tempo range around 90 bpm and started to get worse as we left this range with the highest error being at the much higher tempo ranges around 150 (Figure 9). What is interesting is that the error is much higher on the high end of the tempos as it is on the lower end of the tempos. What we found is that the octave error of the highest bpm is more pronounced than in any other songs. What this could potentially mean is that there is a participant that is experiencing the 150 bpm song as a half octave down or at around 75 bpm. The issue that this also plays on our results is that it adds more data to the middle where the model is going to see that more of the data still belongs within that range, possibly causing it to over emphasize that region of high density.

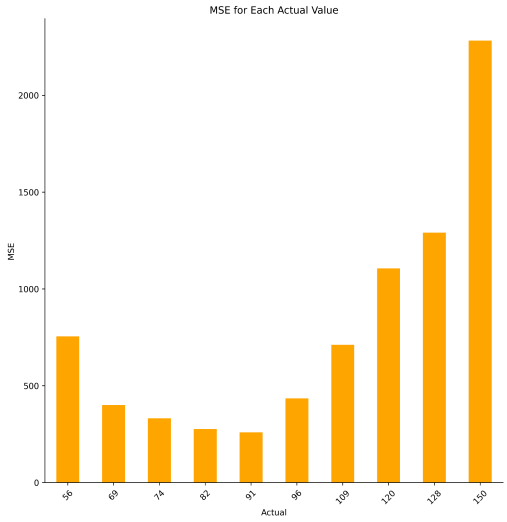


Figure 9. Bar graph plot showing the mean squared error (MSE) for each class.

4.3. 1D CNN for multiclass classification results

Classification yielded satisfactory results, with the best accuracy being 87.5% for the 3-class model and 85% for the 5-class model. It makes sense that the 3-class model would perform better, since the prediction buckets are larger and there are less of them. Besides printing the accuracy metric, confusion matrices were used to visualize model performance by plotting the counts of correctly predicted classes and falsely predicted classes for each class as can be seen in Figure 10.

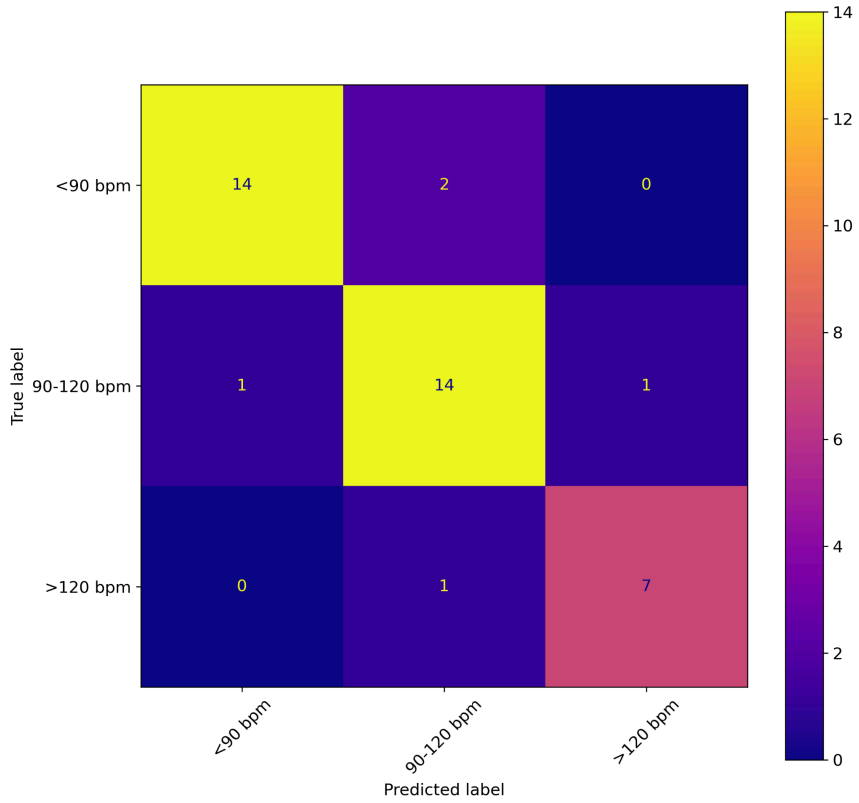
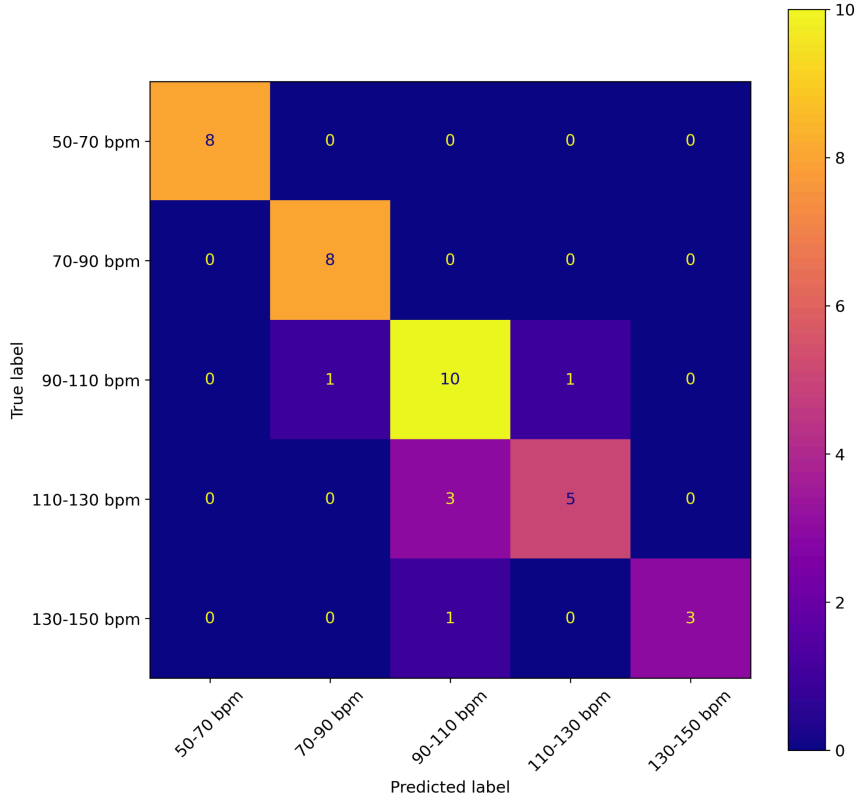


Figure 10. Confusion matrix for the classification model with 3 classes (top) and 5 classes (bottom).



Whether due to the complexity of the chosen architecture, the small size of the dataset, or other factors, both classification models struggled with overfitting, and would converge very rapidly as can be seen in Figure 11.

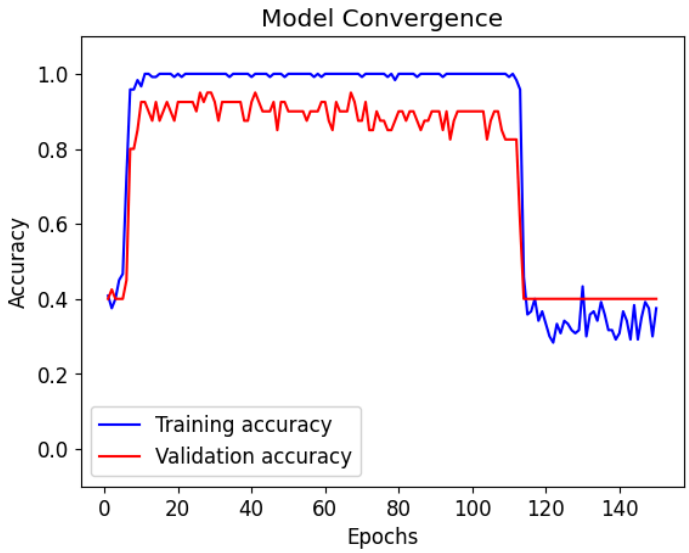


Figure 11. Model convergence for classification with 3 classes.

4.4. Challenges

We encountered three main challenges throughout the project that manifested themselves in our results: (1) There was a higher density of data around the 90 bpm tempo range and our models experienced bias towards predicting closer to these values or had better accuracy when predicting on samples within this range. Using class weighting schemes proved to mitigate this in the classification approach. (2) For very fast and very slow tempos the models began suffering from the effect of octave error. In this context, octave error occurs when a person perceives tempo as being at an integer multiple or fraction of the tempo that the song is listed at, which occurs more often with very slow or very fast tempos [7]. If while listening a participant's brain synchronizes to a tempo an octave removed from the actual tempo, it renders the target values and labels for that sample incorrect, which makes the model struggle to make accurate predictions. Because human brains tend to favor synchronizing with middling tempos rather than very slow or fast tempos, octave error also contributes to the higher prediction density around 90 bpm. (3) Our dataset was large in size (30'000 data points per sample) but small in song variety, so our models were prone to overfitting despite measures taken to counteract it.

5. Future Work

To improve the performance of the models and our understanding of the data, we would like to explore the following in future work: (1) Creating a two-feature model by fusing both the time series data and the frequency domain of the EEG data could help the model hone in on a prediction better. (2) In our regression models we would like to account for the higher density of data in the middling tempo ranges using a weighting technique, as this might reduce the error for slow or fast tempo songs. (3) Making the loss invariant to the octave error would be a more realistic way to estimate tempo and is easier to accomplish than identifying and accounting for octave error in participant perception. With this change, the models would accept tempos predicted at half or double the actual tempo as a correct prediction. (4) Changing the target value from beats per minute (bpm) to beats per second (Hz). (5) Adapt the models to work on other datasets and incorporate transfer learning.

6. Conclusion

In this project we created three different deep learning models to test our hypothesis that machine learning can be applied to EEG data to find features in the data that can be used to determine song tempo. The first, and simplest, model was a multilayer perceptron that implemented multiple linear regression. This model performed the worst out of all the models, but it required the least training time. The second model was a 2D convolutional neural network (CNN) for regression. This model performed well around the areas of high density and poorly outside of this. Using K-fold cross validation across the 10 songs showed this was the case when using the outer songs as validation as opposed to the inner songs. The third model we created was a 1D CNN for multiclass classification which yielded satisfactory results at 87.5% accuracy with 3 classes and 85% accuracy with 5 classes.

Each one of our models showed the same tendency to predict closer to the median of all the song tempos. This resulted in higher accuracies in that tempo range and much worse accuracies in the lower and higher tempo ranges. This problem was caused in part because of the nature of our dataset as well as a phenomenon called octave error. The first part of this problem could be addressed in future work by using larger data sets, or by using more data sets in addition to the NMED-T data set we used in this project. The octave error portion of this problem could be addressed by making the loss invariant to the octave error. Despite these issues, we have shown that this method of tempo estimation is valid. With some future work and problem solving, this method could become accurate enough to be used in real world applications.

7. Acknowledgements

We would like to acknowledge our faculty advisors Dr. Blair Kaneshiro and Dr. Masoumeh Heidari who brought this project to our attention and guided us throughout the entire process. Dr. Kaneshiro is a neuroscience researcher at Stanford who specializes in auditory signal processing and EEG applications. She was part of the group that created the NMED-T dataset that we used in our project [3]. Because of this, she was an excellent resource for information on the data we were working with. Dr. Heidari is an assistant professor at University of Alaska Anchorage, and a researcher focused on machine learning and artificial intelligence. She provided guidance when we were designing our models and answered many questions we had on complicated machine learning topics. Thank you Dr. Kaneshiro and Dr. Heidari for your contribution and guidance.

9. References

[1] Schreiber, H., Urbano, J., & Müller, M. (2020). Music tempo estimation: Are we done yet?. Transactions of the International Society for Music Information Retrieval, 3(1).

[2] Stober, Sebastian, Thomas Prätzlich, and Meinard Müller. "Brain Beats: Brain Beats: Tempo Extraction from EEG Data." Proceedings of the Proceedings of the International Conference on Music Information Retrieval (ISMIR) New York, USA, 2016. 276-282.

[3] Losorelli, S., Nguyen, D. T., Dmochowski, J. P., & Kaneshiro, B. (2017). Naturalistic music EEG dataset - tempo (NMED-T). Stanford Digital Repository. https://purl.stanford.edu/jn859kj8079

[4] Losorelli, S., Nguyen, D. T., Dmochowski, J. P., & Kaneshiro, B. (2017, October). NMED-T: A Tempo-Focused Dataset of Cortical and Behavioral Responses to Naturalistic Music. In ISMIR (Vol. 3, p. 5).

[5] Dmochowski, J. P., Greaves, A. S., & Norcia, A. M. (2015). Maximally reliable spatial filtering of steady state visual evoked potentials. *NeuroImage*, *109*, 63–72. https://doi.org/10.1016/j.neuroimage.2014.12.078

[6] Zhang, J., Liu, D., Chen, W., Pei, Z., & Wang, J. (2022). Deep convolutional neural network for eeg-based motor decoding. Micromachines, 13(9), 1485.

[7] Elowsson, A., & Friberg, A. (2015). Modeling the perception of Tempo. *The Journal of the Acoustical Society of America*, *137*(6), 3163–3177. https://doi.org/10.1121/1.4919306

Appendix A: Requirements

Since this was a research project without a client stakeholder, no formal requirements had been set forth for us. We came up with the following requirements to guide our process. These requirements have been informally adjusted multiple times throughout the duration of the project.

1. Functional Requirements
    a. The machine learning model must be able to ingest the preprocessed EEG data of an static agreed-upon or dynamically changeable size.
    b. Data must be split into test, training, and validation sets that maintain an equal distribution of songs.
    c. The best-result models must be able to predict the tempo of the songs that form part of the validation set data to a certain degree of accuracy.
    d. Informative data visualizations must be created to evaluate the performance of a trained model using matplotlib.
    e. The best-result models must be saved as a .keras file, along with a summary of model performance metrics and accompanying visualizations.

2. Non-functional Requirements
    a. EEG data files must follow an agreed-upon file naming convention and row/column structure and naming convention.
    b. Source code must contain concise comments that explain the why as well as the what of the code.
    c. Use docstrings to document functions and classes within the code according to PEP 257 standards.
    d. A trained model should be able to perform a prediction within negligible time to be considered successful.

3. System Specifications

    a. Programming Languages
        i. Matlab will be used to perform the signal processing part of the data preprocessing.
        ii. Python will be used to perform the remaining data manipulation and program the machine learning models using the following libraries:
            1. scikit-learn
            2. keras

    b. Hardware
        i. Google Colab paid-for hardware options will be used to train the machine learning models:
            1. NVIDIA V100 GPU
            2. NVIDIA T4 GPU
            3. High RAM environment

    c. Team Collaboration
        i. Google Drive will be used for file sharing and team collaboration on deliverables.
        ii. Discord will be used for daily communication and asking/answering questions.
        iii. Zoom will be used for weekly team meetings with the faculty advisors.
        iv. Google Colab will be used for programming collaboration.

Appendix B: Software Development Process

At first we created one very simple model to ensure that all the data preprocessing steps were working correctly. We then decided to split up and create separate models based on that first simple model. The development of these models was iterative in nature. We would try an architecture, test the model's performance, then tune the hyperparameters or switch to a different architecture. There were many times when we wanted to implement an advanced technique that we weren't familiar with. We would need to research this topic for a while, until we understood how it would affect our model, and then figure out how to implement it into our model. Because this was a research project, there weren't any specific performance requirements. Our goal was to gradually improve our models to see how accurate we could make them before our deadline arrived.

Planned project timeline:

	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13	Week 14	Week 15
Research, visualize and further preprocess data (MATLAB)												
Port data into Python-ingestable format												
Split dataset into training, validation, and testing sets												
Research and choose appropriate ML algorithm												
Implement the ML algorithm using appropriate Python library												
Train the chosen model, apply cross-validation techniques												
Testing of the model and hyperparameter tuning												
Research evaluation techniques, model evaluation												
Code review, code cleanup, final presentation												

Appendix C: Actions Taken From Code Inspection

The code inspection identified 20 defects in our submissions. We addressed each defect as follows:

1. Naming conventions vary:

This defect was considered a non-issue and nothing was changed. Our code uses the standard scientific notation for some variables which comes off as inconsistent but is actually more clear for someone who is familiar with our topic and its related fields.

2. Repetitive code for removing song:

This defect was considered a non-issue and nothing was changed. In the interest of time, and because the defect wasn't too extreme, we decided to leave the code as it was.

3. Replace generic exception with specific one; consider retaining generic:

A specific exception handler for a FileNotFound error was implemented while retaining the generic exception handler.

4. Naming convention to differentiate objects:

That section of code used a variable of the name "history" which has a commonly used "history" method attached to it. The variable name was changed to model_history for better readability.

5. Combine if IN_COLAB blocks:

These separate if statements with the same condition were joined.

6. Why catch an exception to raise another exception:

We added a print statement inside of the exception catch to add a bit more information for debugging before raising the error again and exiting the program.

7. Long data. reference line could be broken up:

This defect was considered a non-issue and nothing was changed. In the interest of time, and because the defect wasn't too extreme, we decided to leave the code as it was.

8. Structure code with functions:

The code for creating box plot diagrams was converted to a function.

9. Consider learn rate decay for training the NN:
This suggestion was considered, but our advisor advised against us using it due to complexity so we didn't implement it.

10. Table & boxplot could use some more detail -- label numbers, add legend:
The boxplot diagram had labels and a legend added to it.

11. Lower case EEGNet variable name:
This defect is similar to #1 and received the same response; nothing was changed.

12. DRY: duplicate code between files:
In order to give each member of the group as much control over their model as possible, we decided to keep our code as it was. Even though much of the code is repeated.

13. Some lines of code are long so maybe split the lines (makes code readable):
The longest lines in the code were broken up and offset for readability.

14. Change how number of participants and song name are entered, not hard coded in:
This defect report did not provide a useful line number so some general elimination of hardcoded values was performed.

15. Magic number, 60 (probably OK):
This defect was considered a non-issue and nothing was changed. In the interest of time, and because the defect wasn't too extreme, we decided to leave the code as it was.

16. Would be easier to visualize the data in a plot instead of a table:
Data visualizations have been created.

17. Y-axis label is vague, "Tempo (bpm)":
Axis label changed to "Predicted beats-per-minute (BPM)". This label is much more descriptive.

18. Magic numbers
This defect was considered a non-issue and nothing was changed. In the interest of time, and because the defect wasn't too extreme, we decided to leave the code as it was.

19. A bit too many comments:

The code was inspected in an unusual format which made the commenting seem extreme. In Google Colab, where the code will usually be used, the commenting is formatted nicely and is very helpful. For these reasons we decided to leave the code unchanged.

20. Can you add the actual tempo for reference to the box plot?:

Lines to denote the target tempo were added to the plot.

Appendix D: Code Listing and Model Usage Documentation

Source code for helper files can be found on our github repo.

- save_song_data_as_csv.py - Eleonora created this file to run a lot of the scripts to aggregate, manipulate, and label the data.
- plot_trial_FD.m, plot_trial_TD.m, run_and_plot_RCA.m, run_rca.m - Files originally from Dr. Kaneshiro's toolbox code that help visualize the data that Eleonora refactored slightly.
- compare_plots.m, retain_uniform_trial_data.m, save_FD_data.m, save_rca_filtered_songs.m - Matlab helper code written by Eleonora to export data to .mat files.

Source code for model setup and training can be found on Google Colab:

- linear_reg_mlp.ipynb - Code that loads in, normalizes, and splits the EEG data before setting up the MLP for multiple linear regression and training the model. Written by Eleonora (50%) with refactoring from Quinton (50%).
- classification_3_classes.ipynb - Code that loads in, normalizes, and splits the EEG data before setting up the 1D CNN and training the model as a 3-class classifier. Written by Eleonora (90%) with refactoring from Jaren (10%).
- classification_5_classes.ipynb - Code that loads in, normalizes, and splits the EEG data before setting up the 1D CNN and training the model as a 5-class classifier. Written by Eleonora (90%) with refactoring from Jaren (10%).
    - To train or run either classification model, it needs to load in a csv file containing the aggregated, RCA-filtered, and uniformly sized data from the NMED-T dataset. The csv file should be located at the file path specified in the code and contain the following columns:

song_name	label	bpm	beat_interval	0	...	29999

The label column represents the class label (0 - 2 for 3 classes and 0 - 4 for 5 classes), and the columns 0 to 29999 represent the total number of data points (30000 data points = 4 minutes of data at a sampling rate of 125 Hz).

In the code, the user can specify the size of the chunks that the data should be split up in during synthetic data augmentation.

- regression_2d_CNN-Code that loads in, normalizes, and splits the EEG data before setting up the 2D CNN and training the model as a regressor. Written by Jaren(95%) Eleonora(5%). The initial start up code was hers. NOTE: Will not run unless using high vram and high ram.
    - To train and run this file you must include all of the imputed.mat files from the NMED-T dataset and change the filepath variable to the file path where these 10 are located.
    - Warning: The NMED-T dataset is very large and can consume around 10 gigs of ram upon loading up.