THE OHIO STATE UNIVERSITY

Music Genre Classifier

Project Category: Audio
Physics 5680, Autumn 2024

Author: Ethan Johnson

December 12, 2024

Abstract

This project is based around a music genre classifying machine learning model. It is motivated by my own personal love for music, and desire to challenge myself to work with audio, something I have never done before. It uses the large Jamendo dataset for model training, and uses a Keras FCN to create the model. The model incorporates multi-label, multi-class classification using the 10 most populated genres in the dataset. The accuracy and loss of the data set were 82%, 1.40 respectively, and also had an AUC of 0.86. Please note that my full length job was unable to finish in time for this paper to be turned in, so all plots and other results are given by a model trained on 500 audio files (350 train, 75 test, 75 split) as opposed to the roughly 8,000 that would have ran.

1 Introduction

For my final project, I have chosen to create a machine learning model that will classify the genre of a given audio file. It is trained on the Jamendo dataset. As someone who has not worked in Python before this semester, it has sure been a challenge, and has tested my skills as a programmer. As a lover of music and musician, I chose to do a project in music as a way to make this project fun and interesting for myself.

In today's world, people have seemingly endless access to music. With this, there are many algorithms used by various streaming services that create "recommended for you" playlists to expose the listener to new artists. This project could act as a sort of building block for those algorithms, as it is able to analyze what genre the given audio track is, and could relate them to various genres and similar types of music. This can be somewhat tedious, as in those audio files it can be difficult to discern what instruments are playing due to the overlap of things such as frequencies and timbres. My model creates four unique features for each audio and combines them into one large array that serves as the basis of the classifier. The Jamendo dataset that I mentioned before consists of four total sets of data - the audio for training and its respective metadata as well as audio and metadata for validation. The metadata contains columns for the ID of the audio file, the artist ID, album ID, duration of the track, genre, instruments, and moods.

The model receives the four unique features (MFCC, a Mel Spectogram, Chroma Vector, and Tonal Centroid Features, which I will explain in Section 3) combined into one unique array. Using Keras, a Fully Convolutional Network (FCN) is used for the algorithm, which will output the predicted genre.

2 Related Work

Many papers and projects have been done on the topic of music genre classification. This is in part due to how common streaming services are used, and that those streaming services use similar algorithms to recommend music to their listeners. Because of this, there are many ways that one can go about creating an algorithm to classify genres, although some of this depends on how your dataset is structured. Some datasets have more information than others, and thus alters how you can approach sorting through the data and ultimately creating your model.

A common approach to genre classification is through the use of deep learning frameworks like Keras, which offers various model architectures. For instance, one approach uses a Fully Convolutional Network (FCN), as demonstrated in a website by Paperspace [4], while another uses a Convolutional Neural Network (CNN), as seen in a paper published by IRJET in 2019 [2]. Both of these models have proven effective in genre classification tasks, with CNNs generally performing well in cases where spatial relationships in data (such as spectrograms of audio signals) are important. However, FCNs can be a better choice for simpler, tabular data where complex feature extraction isn't needed. In my case, the feature extraction didn't get too complex, as I only extracted four features.

Another popular approach is using Random Forests with libraries like XGBoost, which are often employed due to their ability to handle large, high-dimensional datasets and their robustness against over fitting. Random Forests are less computationally expensive than deep learning models and can be more interpretable, but they might not capture complex patterns in data as effectively as CNNs or FCNs. Still, for certain applications where interpretability and speed are critical, Random Forests can be a very competitive option.

I came across a helpful guide that provides an overview of various models for music genre classification by a blog cite called Medium [3]. While the dataset it uses differs from mine, the comparisons of accuracy across different models are insightful. The guide highlights that CNNs are often regarded as the current state-of-the-art for this task. However, many of the papers suggesting this are a bit dated, with newer advancements in the field potentially improving upon these methods [5].

In this project, I am exploring a deep learning Fully Convolutional Network by Keras, although I did attempt to create a model using XGBoost that did not perform nearly as well as the FCN. Although the XGBoost model is very adaptable and versatile, it was not as adaptable as the FCN, as it is more suited for structured data rather than something such as audio. Although the FCN is perhaps more computationally intensive, it in fact did produce better results all across the board.

3 Dataset

The data I will be using is the Jamendo dataset [1]. The dataset contains over 55,000 full audio tracks (in the form of .opus files) with 195 tags from genre, instrument, and mood/theme categories, and is also split into 90% train and 10% validation. The dataset also has 2 folders containing the respectively split data, as well as 2 .tsv files that contain all the metadata. The metadata contains columns for the ID of the audio file, the artist ID, album ID, duration of the track, genre, instruments, and mood. To work with this data, I have created five functions that compute unique features for each audio file and stores them in an array with length of 498. With this, all the arrays become normalized to the same length.

There are many audio tracks in the .tsv files that contain the metadata that is empty, and have been removed from my data used in modeling. In order to get the full train/test/val split, I used the function train_test_split from the scikitlearn [6] package to split the data into 80% training and 20% testing, leaving the full dataset to be 72% training, 18% testing, and 10% validation. For cleaning, I chose to use only the 10 most populated genres, and after cleaning the dataset, I was left with approximately 8,000 audio files for training, and 2,000 for testing and validation (split 50/50).

While analyzing the data, I found an that the genres in both the training and validation sets are proportional in the distribution of types of genres, see Figures 1 and 2. This makes the above splitting the data convenient and easy to work with when creating the model. A crucial part of using the data is by extracting features of the data, as I mentioned in Section 1. The first feature is just simply a waveform of vs time graph that

shows how loud or soft the audio file is throughout the duration of the song. The second feature is called the Mel-Frequency Cepstral Coefficients (MFCC), which are derived from a Fourier transform of an excerpt of a signal, then mapping the powers of the spectrum onto the Mel Scale (a logarithmic scale of pitches that models how we perceive sound). It then takes the logs of the powers at each of the mel frequencies, which are then used in a discrete cosine transform. The amplitudes of this are then used for the resulting spectrum seen in Figure 3. The third feature is called a Mel Spectogram, and is a visual representation of an audio signal's frequency spectrum over time through a Fourier transform to break down the signal (Figure 4). The fourth feature is called a Chroma Vector, and is created by having the full spectrum projected into 12 bins that correspond to a musical octave (Figure 5). The final feature builds off the Chroma Vector, and is called the Tonal Centroid Features (Tonnetz for short). Tonnetz is created by projecting the Chroma Vector onto a 6-dimensional basis that represent the perfect fifth, minor third, and major third as two dimensional arrays (Figure 6).

4 Methods

To create my algorithm, I use Keras to create a Fully Convolutional Network. Throughout all of my trials I have used some form of an FCN, partially because my embeddings are done in the form of a 1-D array, which FCNs are designed to work with. I had considered using a package called XGBoost (Extreme Gradient Boosting), but chose to stick with something I was familiar with, as XGBoost gave me some issues. The model is somewhat simple consisting of only three dense layers and a dropout layer. Most of my problems throughout this project were surrounding the data itself, mostly the classification of the genres due to the amount of genres present in the data (256 total). My methods involved basic single-label classification, multi-class classification, and multi-label classification.

I began this project attempting to clean the data in a way that I could create an early model. This model was just to show that the cleaned data was able to be used in a machine learning manner. I used a simple binary classifier that was able to classify only electronic and classical music genres. The model that I used for this was just a simple Keras FCN, much like what I describe below in Section 4.2. I later attempted to use multi-class classification, where I assigned the top 10 most populous genres 0-9 and one hot encoded them as my labels. This caused issues however, as the Jamendo dataset included multiple genres in a large number of the songs. When I followed through with my embeddings and went to train and run the model, that I was getting very bad performance, no matter how I adjusted the data. I then tried something I have not used before, that being something resembling multi-label and multi-class classification. I did this by doing as I had done before, and filtered out any audio file that had a genre outside of the top 10 most populated (electronic, soundtrack, ambient, pop, experimental, classical, easylistening, rock, alternative, and chillout). But this time, I allowed for multiple genres to overlap, and one hot encoded the genres into an array of length 10, with each column corresponding to a specific genre. When I went through with the embeddings, it resulted in an accuracy of 82% and loss of 1.40.

4.1 Model 1 - XGBoost

XGBoost is a machine learning algorithm that takes some concepts that we have worked on in class to a new level. Similarly to random forests, XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. This is done by creating decision trees sequentially (decision trees are a supervised learning algorithm that uses a tree-like structure to make predictions by asking a series of questions based on data features, where each node represents a decision point, and the final outcome is determined by following a path through the tree to reach a leaf node). Weights are then assigned to all independent variables and sent through the decision trees. It then takes any variables wrongly predicted and weighs them heavier before sending them through a second decision tree to learn from its mistakes, and so on and so forth. This would have been great to implement into my code, as it is highly compatible with a wide variety of tasks, but instead I chose to work with what was familiar (my accuracy was also somewhere around 50% while using it).

4.2 Model 2 - Keras FCN

Throughout this course, we have become very familiar with using Keras as our go-to machine learning library. Because of that, as well as the numerous papers I skimmed through that pointed to this being the right model to choose, I chose to create a FCN using Keras. The model, as I mentioned earlier, is fairly simple, and consists of 3 "dense" layers and 1 "dropout" layer, as well as its inputs and outputs ("Dense" is a linear operation in which every input is connected to every output by a weight, followed by a nonlinear activation function, and "dropout" simply halves the number of nodes used throughout the training to help prevent over fitting). The inputs for this model were simply the one-dimensional array for each unique song, and the outputs were the predicted genre of that song. The dense layers compress from size 256 down to 64. The model is then compiled using "Adam" as its optimizer with learning rate 1e-4. The "Adam" optimizer is somewhat complex, but in short helps push the neural network along in learning more efficiently and effectively. And lastly, my callbacks use EarlyStopping to help prevent the model from over fitting. This is done by keeping track of the "val_loss", in this case the predicted loss, so that when it begins to increase after decreasing as it should it stops the training. The performance of the model is then saved into the dataframe "history", and two graphs are shown depicting accuracy and loss for both the testing and validation against the epochs to show performance of the model (see Figures 8 and 9).

5 Results/Discussion

For my algorithm's hyper-parameters, I chose them by trial and error to see what gave me the best results, just tweaking them until I got the highest accuracy. For my input, I chose a 1-D array, as that was best for my data and the model. I have 3 hidden layers, as I stated in Section 4. I attempted to use more, and condense the nodes down further but struggled with over fitting when I did so. For my activation functions, I went with using ReLU for the dense layers (we used this the most in class, so I figured it would do the job), and sigmoid for the output layer, as it works well when doing multi-label classification (sigmoid assigns a probability of the label being true for each label). Again, for optimization I went with what I have become accustomed to and used Adam with a learning rate of 1e-4. I branched out into something new using BinaryCrossentropy as my loss function because it works well with one hot encoded multi-label classification. I did not choose a batch size, as I thought that the 32 default did well enough, and also chose 64 epochs to run on. I used early stopping, which monitors "val_loss" with a patience of 10 epochs (i.e. training is stopped after 10 epochs if training shows no improvement). My model did not use any sort of cross validation, as my data was split into train, test, and split using sklearns "train_test_split" function.

I chose Accuracy and AUC as my metrics because together they give a balanced picture of how well my model is performing. Accuracy is a straightforward measure of whether the model is correctly predicting genres. It is calculated as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Predictions}}$$

While it provides an intuitive sense of overall performance, it can be misleading in cases where the data is imbalanced. This is because accuracy does not account for false positives/negatives, which may disproportionately affect certain labels or genres. This was a huge issue I had when I first tried to do multi-label classification, because there is a large disparity between the most and least populated genres, as my accuracy was very high, and my AUC among other metrics were very low. This led me to instead using only the top 10 populated genres, so that although there still might be some degree of disparity, it still leads to sensible results from the model.

AUC (which stands for Area Under the Curve) is given by :

$$\text{AUC} = \int \text{ROC curve}$$

It is on a scale of 0-1, with 0.5 being the worst baseline (random guessing), and 1 meaning classification is perfect, so we want to be as close to 1 as possible. For my data, I had an AUC of 0.86, which is pretty good.

Some other metrics I used included precision, recall, and F1 score (both micro and macro for all). Micro averages are based on the global counts and give more weight to classes with more instances, while macro treats all classes equally regardless of size. Precision focuses on avoiding false positive, recall avoids false negatives, and F1 score is a balance between the two. They are given by the equations below.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{F1 score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

I use those metrics to expand my view of how well my model is performing, and it does indicate some flaws, specifically on the macro scale.This could be stemming from an imbalance of the genres, leading to an under representation of less frequent genres. I also stumbled across a metric which I hadn't heard of before called **hamming loss**. Hamming loss is good at evaluating the performance of multi-label classification, as it measures the fraction of incorrect labels predicted by the model compared to the total labels. It is the opposite of AUC, as you want to be as close to 0 as possible. The formula is given below :

$$\text{Hamming Loss} = \frac{1}{N \times L} \sum_{i=1}^{N} \sum_{j=1}^{L} \mathbf{1}(y_{ij} \neq \hat{y}_{ij})$$

I do not have any data on the XGBoost model I attempted, as I abandoned it before I got to that point due to already having success with the Keras FCN. Unfortunately, I was also struggling to get graphs for the other 4 metrics I used, so I only have graphs for Accuracy and Loss, shown below (Figures 8 and 9). Below are two tables of the final results.

Accuracy	Loss	AUC	Hamming Loss
0.82	1.40	0.86	0.043

Table 1: Final accuracy, loss, AUC, and Hamming loss. All values suggest that the model is performing well, and that it is correctly predicting most genres.

F1 Score	Precision	Recall	
0.68	0.78	0.61	Micro
0.08	0.081	0.078	Macro

Table 2: Final F1 score, precision, and recall for both micro and macro. The micro results suggest that the model is performing well in classes that dominate the dataset. However, the macro shows that the model is struggling with certain classes. This could be due to class imbalance, as I was unable to run the full dataset.

6 Conclusions/Future Work

This project has been a learning experience for me, as I have never created a project from scratch. Using various techniques we learned in class, as well as some things I by myself along the way, I was able to filter the large Jamendo dataset to create a multi-label classifier of 10 music genres. The model I used was a Keras Full Convolutional Network, that overall does a good job at classifying (see the tables in Section 5 as well as Figures 8 and 9). Although I didn't expand too far out into using other models due to my own personal limitations and time constraints, of the two I did test (XGBoost and Keras FCN), the FCN greatly outperformed the XGBoost model, although I'm not exactly sure as to why. If I were given more time, as well as a group to work with, I certainly would have explored other models, as I found a multitude while

doing research for this project (CNN, XGBoost, Random Forest). I would also attempt to dig a little deeper into any overfitting issues that I am sure I have in the model, as well as extend the number of genres that the model can classify.

7 Figures

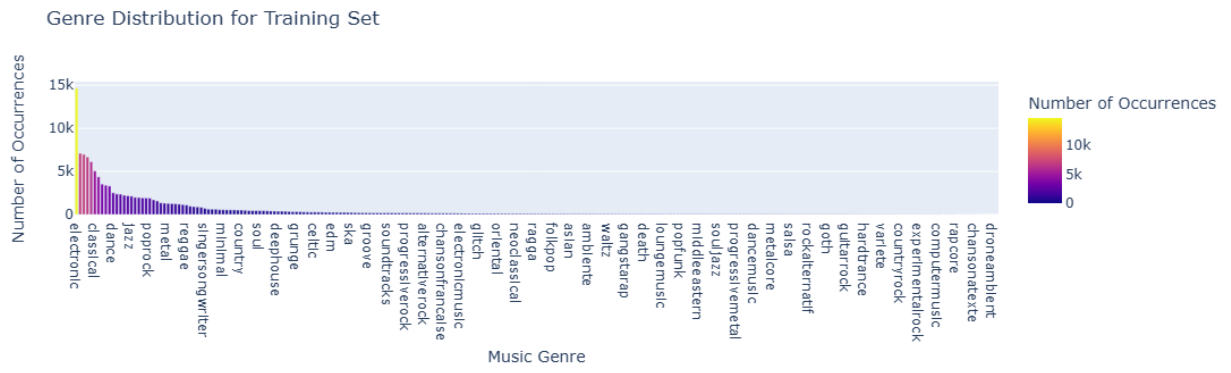

Figure 1: Occurrences of each genre in the training set. This figure shows the distribution of genres in the training dataset, highlighting how frequently each genre appears.

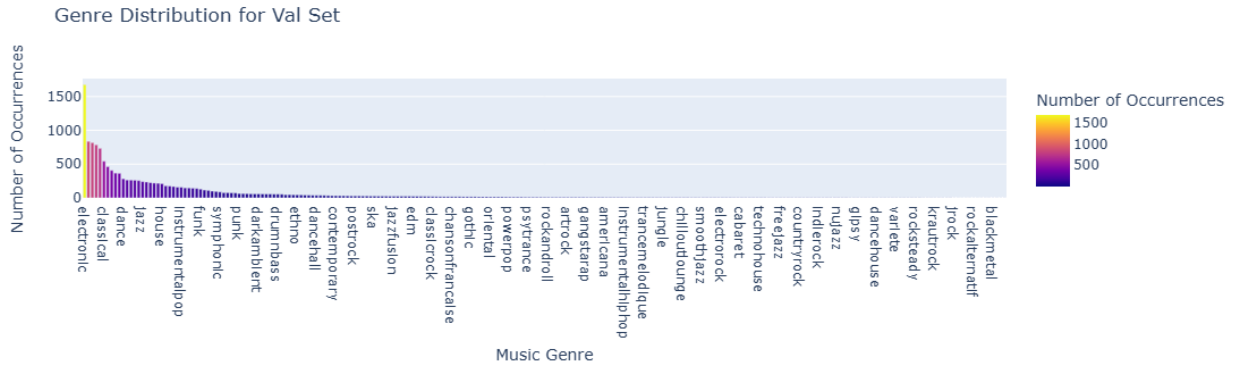

Figure 2: Occurrences of each genre in the validation set. Similar to the previous figure, this distribution shows how genres are represented in the validation set. Note the proportionality to Figure 1.

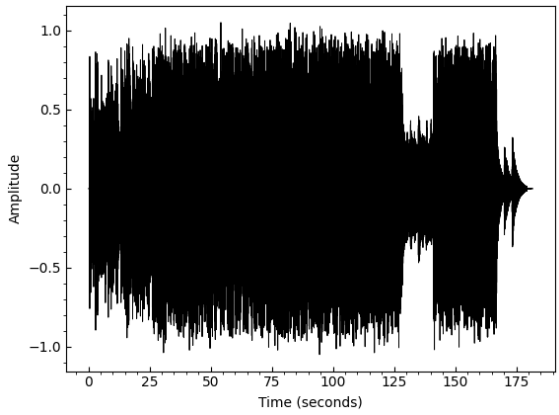

Figure 3: Waveform of the audio file 1234446.opus over time. This figure illustrates the raw audio signal as a function of time. It shows how the amplitude of the signal varies, which can provide insights into the structure and rhythm of the audio, although more advanced features like spectrograms or MFCCs are typically used for classification tasks.

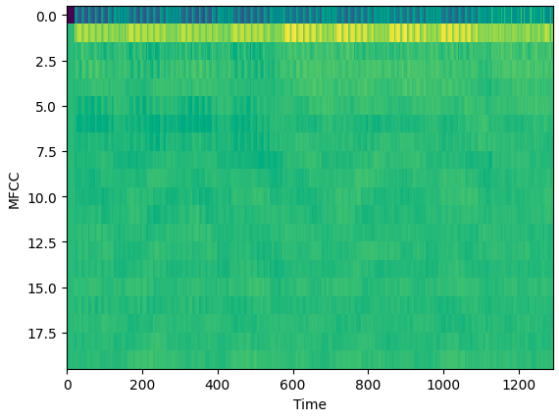

Figure 4: Mel-Frequency Cepstral Coefficients (MFCC) for audio file 1234446.opus. MFCCs are widely used in speech and music classification tasks as they represent the short-term power spectrum of the sound, providing a compact and informative representation of the audio signal. This feature is commonly used to capture timbre and spectral characteristics.

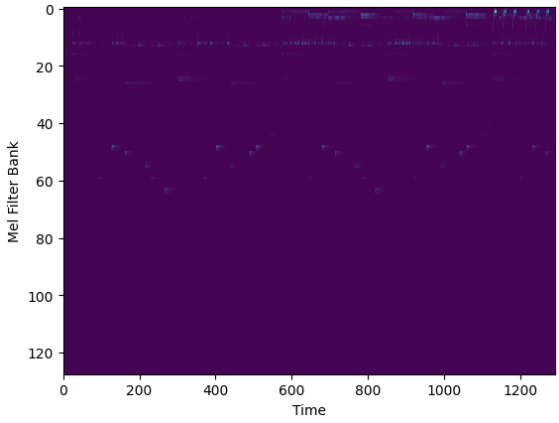

Figure 5: Mel Spectrogram of audio file 1234446.opus. This figure displays the Mel spectrogram, a transformation of the audio signal that maps it into a time-frequency domain. The Mel scale emphasizes the frequencies that humans perceive most acutely, making this feature particularly useful for music and speech classification tasks

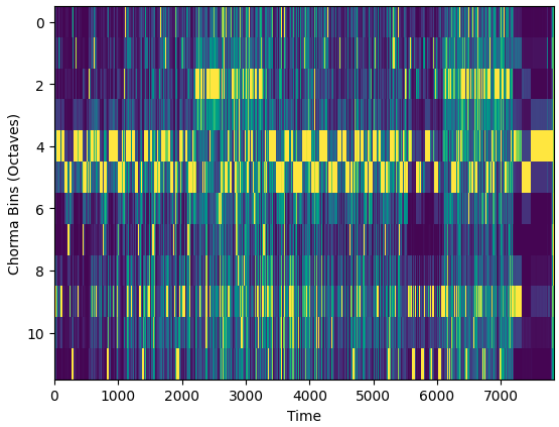

Figure 6: Chroma Vector of audio file 1234446.opus. The Chroma vector captures harmonic and melodic content in music, specifically focusing on the twelve different pitch classes. This feature is useful for tasks like genre classification, as it reflects the musical harmony and key structure of the audio

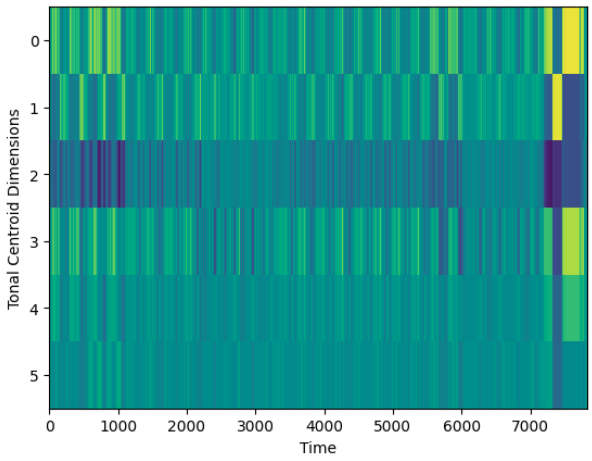

Figure 7: Tonnetz for audio file 1234446.opus. The Tonnetz is a feature used in music analysis that captures harmonic relations and tonal characteristics within the audio. It provides insights into the harmonic progression and emotional content of the music.

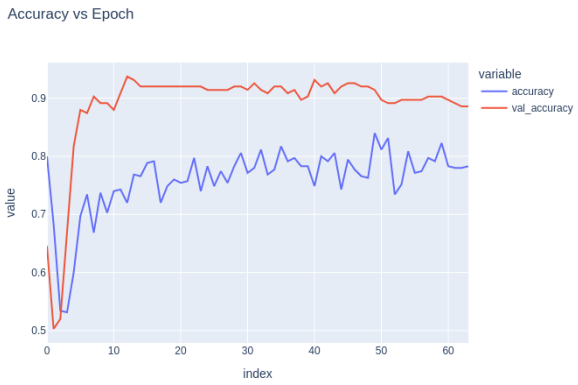

Figure 8: Accuracy improves over time for both training and validation sets, showing that the model is performing well.

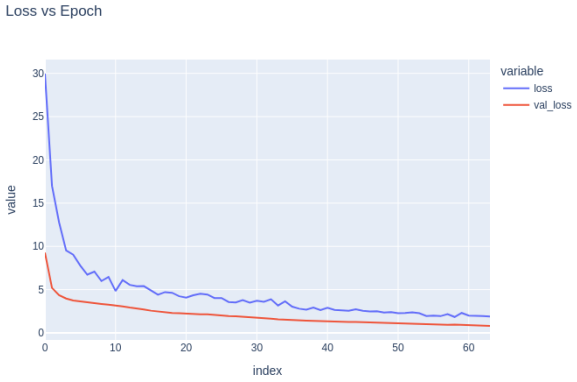

Figure 9: Loss improves over time for both training and validation sets, showing that the model is performing well. We do see that it creeps back up after about the 25th epoch, which could mean the model is slightly overfitting.

References

[1] Bogdanov, D., Won M., Tovstogan P., Porter A., and Serra X. (2019). The MTG-Jamendo Dataset for Automatic Music Tagging. Machine Learning for Music Discovery Workshop, International Conference on Machine Learning (ICML 2019).

[2] Chillara, S., A, K. A., Neginhal, S. A., Haldia, S., and S, V. K. (2019, May). Music Genre Classification using Machine Learning Algorithms: A comparison . www.irjet.net. https://www.irjet.net/archives/V6/i5/IRJET-V6I5174.pdf

[3] Leonhardt, J. (2021). Music Genre Classification: A Machine Learning Exercise. Medium. Retrieved from https://medium.com/@juanfraleonhardt/music-genre-classification-a-machine-learning-exercise-9c83108fd2bb

[4] Paperspace. (2023). Music Genre Classification Using Librosa and PyTorch. Retrieved from https://blog.paperspace.com/music-genre-classification-using-librosa-and-pytorch/

[5] Papers With Code (2024). https://paperswithcode.com/task/music-genre-classification

[6] API design for machine learning software: experiences from the scikit-learn project, Buitinck et al., 2013.

[7] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[8] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." In Proceedings of the 14th python in science conference, pp. 18-25. 2015

[9] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.

[10] McKinney, W., and others. (2010). Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference (Vol. 445, pp. 51–56)

[11] Plotly Technologies Inc. Collaborative data science. Montréal, QC, 2015. https://plot.ly