AI-ML PROJECT : MUSIC GENRE CLASSIFICATION USING CNN

Aanya Verma-200050001
Durgam Latha-200050037
Kumari Pragati Gupta-200050063
Mukkara Gracy paul-200050079

CONTENT

1.INTRODUCTION

We have all used some music apps like Spotify and Billboard, in which we get recommendations to listen to songs. In Music apps, the app recommends playlists of different songs based on our taste in music and also it gives us a list of playlists on a particular genre according to our choice.

In this project “Music genre classification”, we are creating a program that predicts the genre of a piece of music. Currently, genre classification is performed manually by humans applying their personal understanding of music. This task has not yet been automated by conventional algorithmic approaches since the distinctions between music genres are relatively subjective and ill-defined. However, the ambiguity of genre classification makes machine intelligence well-suited to this task. Given enough audio data, of which large amounts can be easily harvested from freely available music online, machine learning can observe and make predictions using these ill-defined patterns.

Music genre classification forms a basic step for building a strong recommendation system. The idea behind this project is to see how to handle sound files in python, compute sound and audio features from them, run Machine Learning Algorithms on them, and see the results. In a more systematic way, the main aim is to create a machine learning model, which classifies music samples into different genres. It aims to predict the genre using an audio signal as its input. The objective of automating the music classification is to make the selection of songs quick and less cumbersome. If one has to manually classify the songs or music, one has to listen to a whole lot of songs and then select the genre. This is not only time-consuming but also

difficult. Automating music classification can help to find valuable data such as trends, popular genres, and artists easily. Determining music genres is the very first step towards this direction.

2.DATASET AND FEATURES

For this project, the dataset that we will be working with is GTZAN Genre Classification dataset which consists of 1,000 audio tracks, each 30 seconds long. It contains 10 genres, each represented by 100 tracks.

The 10 genres are as follows:

1. Blues
2. Classical
3. Country
4. Disco
5. Hip-hop
6. Jazz
7. Metal
8. Pop
9. Reggae
10. Rock

The dataset has the following folders:

- **Genres original —** A collection of 10 genres with 100 audio files each, all having a length of 30 seconds (the famous GTZAN dataset, the MNIST of sounds)
- **Images original —** A visual representation for each audio file. One way to classify data is through neural networks because NN's usually take in some sort of image representation.
- **2 CSV files —** Containing features of the audio files. One file has for each song (30 seconds long) a mean and variance computed over multiple features that can be extracted from an audio file. The other file has the same structure, but the songs are split before into 3 seconds audio files.

The datasets used for following methods are

1. CNN: The features drawn from both image and audio datasets are used in this method.
2. KNN: The features drawn only from audio dataset are used in this method.

We also experimented with pre-processing our data by converting the raw audio into mel-spectrograms. In doing this, we experienced significant performance increases across all models. Mel-spectrograms are a commonly used method of featurizing audio because they closely represent how humans perceive audio (i.e. in log frequency). In order to convert raw audio to mel-spectrogram, one must apply short-time Fourier transforms across sliding windows of audio, most commonly around 20ms wide. With signal x[n], window ω[n], frequency axis ω, and shift m, this is computed as

$$\mathbf{STFT}\{x[n]\}(m,\omega) = \sum_n x[n]w[n-m]e^{-j\omega n}.$$

These are computed more quickly in practice using sliding DFT algorithms. These are then mapped to the mel scale by transforming the frequencies f by

$$m = 2595\log_{10}(1+\frac{f}{700}).$$

Then we take the discrete cosine transform of the result (common in signal processing) in order to get our final output mel-spectrogram. In our case, we used the Librosa library [5] and chose to use 64 mel-bins and a window length of 512 samples with an overlap of 50% between windows. Based on previous academic success with such transformations, we then move to log-scaling using the formula log(X^2). The resulting data can be visualized below:

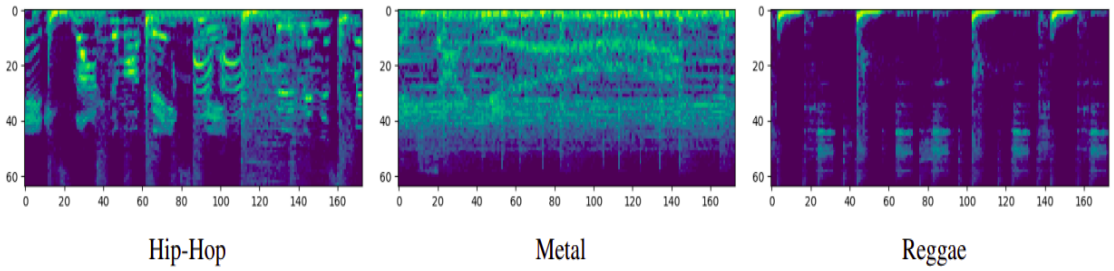


Figure 1: Examples of log-scaled mel-spectrograms for three different genres.

2.1. LIBRARIES USED IN THE PROGRAM

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy
import sys
import os
import pickle
import librosa
import librosa.display
from IPython.display import Audio
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import tensorflow as tf
from tensorflow import keras
```

Libraries used for understanding audio files:

LIBROSA: Librosa is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems. By using Librosa, we can extract certain key features from the audio samples such as Tempo, Chroma Energy

Normalized, Mel-Frequency Cepstral Coefficients, Spectral Centroid, Spectral Contrast, Spectral Rolloff, and Zero Crossing Rate.

IPython.display.Audio: It is a library used for playing the audio in the jupyterlab.

The libraries used for following methods are

3. CNN: Librosa library is used in this method.
4. KNN: python_speech_features library is used in this method.

2.2. FEATURES

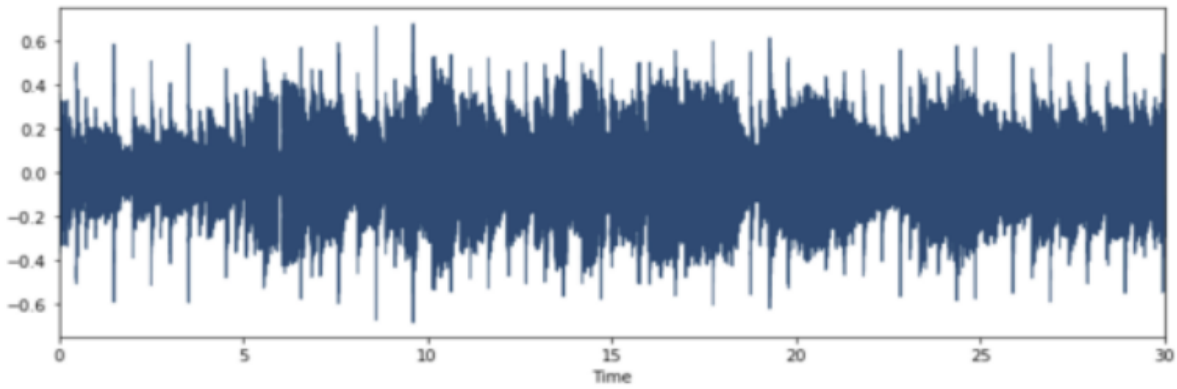
The following methods are used for visualizing audio files we have:

1. Plot Raw Wave Files:

   Waveforms are visual representations of sound as time on the x-axis and amplitude on the y-axis. They are great for allowing us to quickly scan the audio data and visually compare and contrast which genres might be more similar than others.

   ```
   plt.figure()
   librosa.display.waveplot(data,color)
   plot.show()
   ```

   The above lines of code is to plot amplitude of sound w.r.t time.



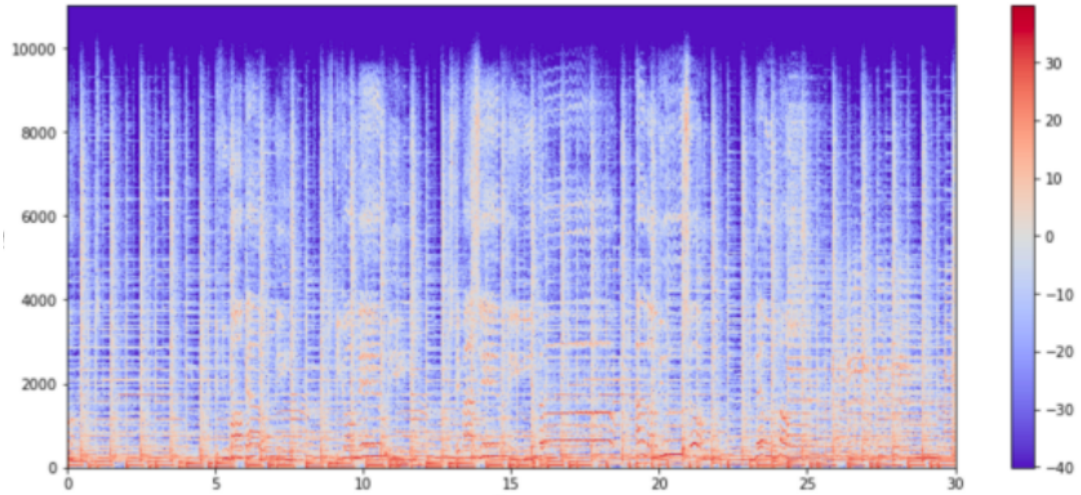
2. Spectrograms:

   A spectrogram is a visual way of representing the signal loudness of a signal over time at various frequencies present in a particular waveform. Not only can one see whether there is more or less energy at, for example, 2 Hz vs 10 Hz, but one can also see how energy levels vary over time.

   Spectrograms are sometimes called sonographs, voiceprints, or voicegrams. When the data is represented in a 3D plot, they may be called waterfalls. In 2-dimensional arrays, the first axis is frequency while the second axis is time.

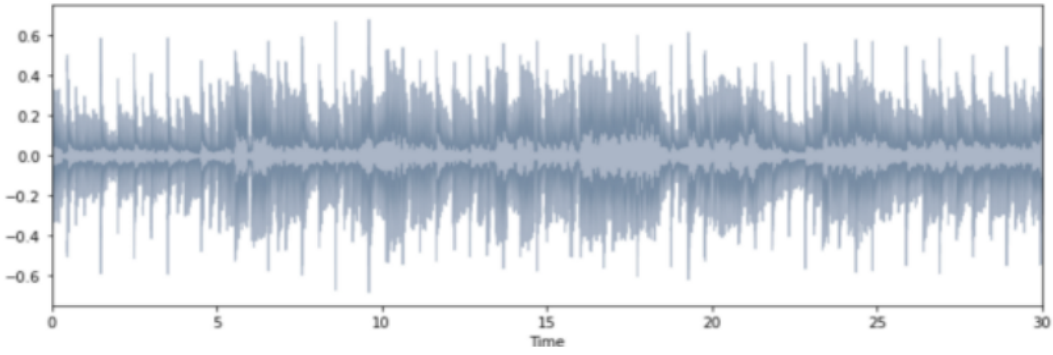
   ```
   stft = librosa.stft(data)
   Stft_db = librosa.amplitude_to_db(abs(stft))
   plt.figure(figsize=())
   library.display.specshow(stft,sr, x_axis, y_axis)
   plt.colorbar()
   ```

   The above code is to plot frequency of sound with time.



3. Spectral Rolloff:

   Spectral Rolloff is the frequency below which a specified percentage of the total spectral energy, e.g. 85%, lies librosa.feature.spectral_rolloff computes the rolloff frequency for each frame in a signal.

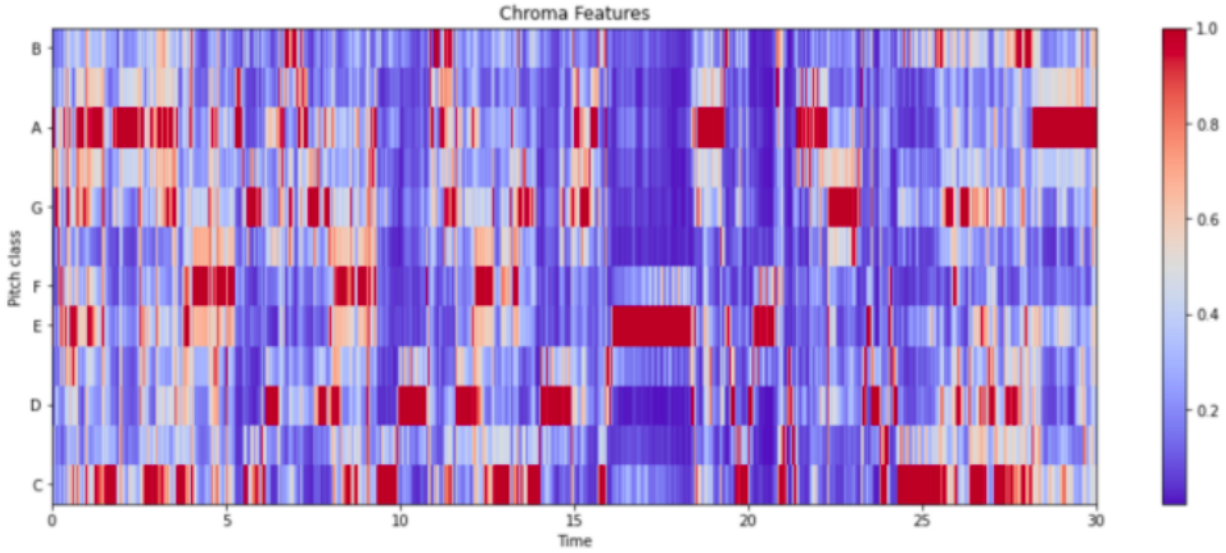


4. Chroma Feature:

   It is a powerful tool for analyzing music features whose pitches can be meaningfully categorized and whose tuning approximates to the equal-tempered scale. One main property of chroma features is that they capture harmonic and melodic

characteristics of music while being robust to changes in timbre and instrumentation.

librosa.feature.chroma_stft computes the pitch class with time.



5. Zero Crossing Rate:

Zero crossing is said to occur if successive samples have different algebraic signs. The rate at which zero-crossings occur is a simple measure of the frequency content of a signal. Zero-crossing rate is a measure of the number of times in a given time interval/frame that the amplitude of the speech signals passes through a value of zero.

librosa.zero_crossings() computes the count of zero crossings in the audio.

2.3. FEATURE EXTRACTION

Preprocessing of data is required before we finally train the data. We will try to focus on the last column that is ‘label’ and will encode it

with the function LabelEncoder() of sklearn.preprocessing. We can't have text in our data if we're going to run any kind of model on it. So before we can run a model, we need to make this data ready for the model. To convert this kind of categorical text data into model-understandable numerical data, we use the Label Encoder class.

2.4. SCALING THE FEATURES

Standard scaler is used to standardize features by removing the mean and scaling to unit variance. The standard score of sample x is calculated as: $z = (x - u) / s$

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data.

3. BUILDING THE MODEL

Now comes the last part of the music classification genre project. The features have been extracted from the raw data and now we have to train the model. There are many ways through which we can train our model. Some of these approaches are

- K-Nearest Neighbors
- Convolutional Neural Networks

3.1. K-NEAREST NEIGHBORS

We know how KNN works by calculating distance and finding the K number of neighbors. To achieve this only for each functionality we will implement different functions.

First, we will implement a function that will accept training data, current instances, and the required number of neighbors. It will find the distance of each point with every other point in the training data and then we find all the nearest K neighbors and return all neighbors. To calculate the distance of two points we will implement a function after explaining some steps to make a project workflow simple and understandable.

Now we have a list of neighbors and we need to find out a class that has the maximum neighbors count. So, we declare a dictionary that will store the class and its respective count of neighbors. After creating the frequency map, we sort the map in descending order based on neighbors count and return the first class.

We also require a function that evaluates a model to check the accuracy and performance of the algorithm we build. So, we will build a function that is a fairly simple accuracy calculator which says a total number of correct predictions divided by a total number of predictions.

you might be thinking like we have implemented the model and now we will extract features from data. So, as we are using KNN Classifier and we have only implemented the algorithm from scratch to make you understand how the project runs, I hope you have a 70 percent idea of how the project will work. So now we will load the data from all the 10 folders of respective categories and extract features from each audio file and save the extracted features in binary form in DAT extension format.

Mel Frequency Cepstral Coefficients

Feature extraction is a process to extract important features from data. It includes identifying linguistic data and avoiding any kind of noise. Audio features are classified into 3 categories high-level, mid-level, and low-level audio features.

- High-level features are related to music lyrics like chords, rhythm, melody, etc.
- Mid-level features include beat level attributes, pitch-like fluctuation patterns, and MFCCs.
- Low-level features include energy, a zero-crossing rate which are statistical measures that get extracted from audio during feature extraction.

So, to generate these features we use a certain set of steps and are combined under a single name as MFCC that helps extract mid-level and low-level audio features. Below are the steps discussed for the working of MFCCs in feature extraction.

1. Audio files are of a certain length(duration) in seconds or as long as in minutes. And the pitch or frequency is continuously changing so to understand this we first divide the audio file into small-small frames which are nearly about 20 to 40 ms long.
2. After dividing into frames, we try to identify and extract different frequencies from each frame. When we divide in such a small frame, assume that one frame divides down in a single frequency.
3. separate linguistic frequencies from the noise
4. To discard any type of noise, take discrete cosine transform (DCT) of the frequencies. Students who are from engineering backgrounds might know cosine transform and have studied this in discrete mathematics subjects.

Now we do not have to implement all these steps separately, MFCC brings all these for us which we have already imported from the python speech feature library. we will iterate through each category folder, read the audio file, extract the MFCC feature, and dump it in a binary file using the pickle module. I suggest always using try-catch while loading huge datasets to understand, and control if any exception occurs.

3.2 FEED FORWARD NEURAL NETWORK

We used a fully connected neural network as well, with ReLU activation and 6 layers, with cross-entropy loss. As the input to our model was 1D, when using mel-spectrograms, we flattened the data. Our model is fully connected, which means each node is connected to every other node in the next layer. At each layer, we applied a ReLU activation function to the output of each node, following the formula:

$$\mathrm{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0. \end{cases}$$

At the end, we construct a probability distribution of the 10 genres by running the outputs through a softmax function:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

To optimize our model, we minimized cross entropy loss:

$$CE(\theta) = -\sum_{x \in X} y(x) \log \hat{y}(x)$$

We experimented with various regularization techniques, such as dropout layers and L2 regularization. Dropout randomly selects features to drop based on a specified constant, and L2 regularization adds a penalty term to the loss function in the form of $\lambda \sum_i \theta_i^2$. This was implemented with TensorFlow.

3.3 CONVOLUTIONAL NEURAL NETWORK

This was our most advanced model, using 3 convolution layers, each with its own max pool and regularization, feeding into 3 fully connected layers with ReLU activation, softmax output, and cross entropy loss. Most of the equations can be found above, and our architecture is visually presented below: This approach involves convolution windows that scan over the input data and output the sum of the elements within the window. This then gets fed into a max pool layer that selects the maximum element from another window. Afterwards, the output is fed through a model described in section 3.1. This was implemented with TensorFlow and Keras.

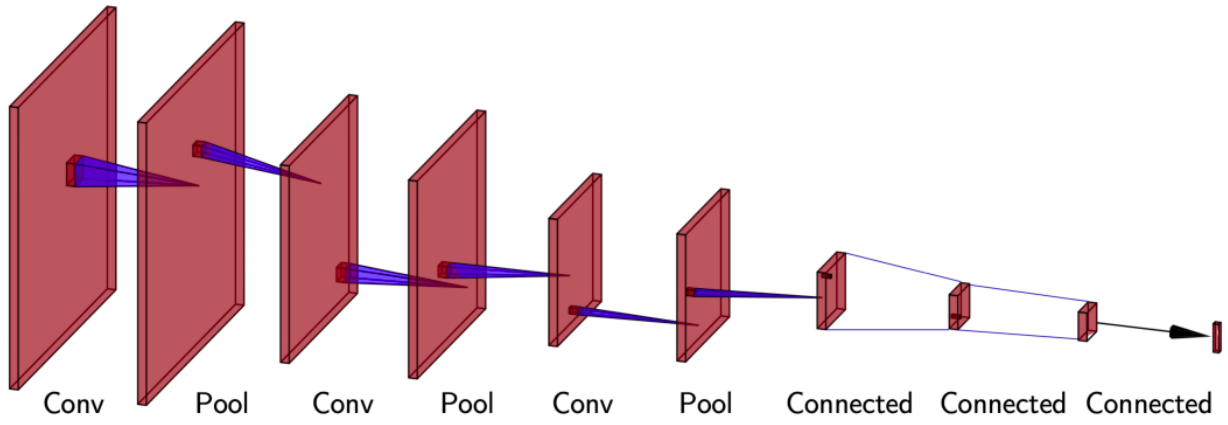


Figure 2: CNN architecture.

4. RESULTS

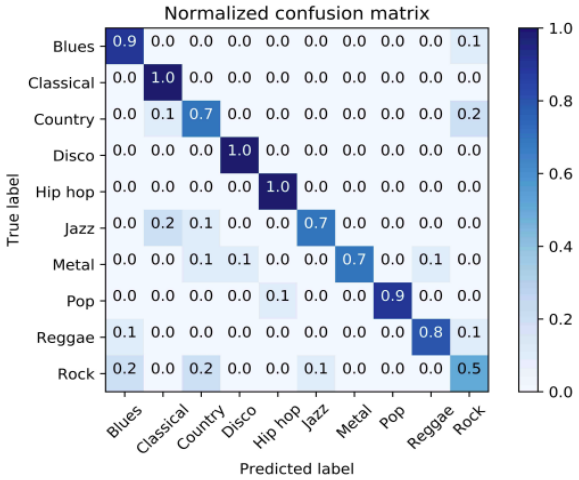


Figure 3: Confusion matrix for CNN predictions.

The main quantitative metric which we used to judge our models is accuracy (that is, percentage of predicted labels which matched their true labels), and our main way of visualizing the performance of our best model is through the confusion matrices as seen in Figure 3. Because the labeling was uniformly distributed on our data, cross-validation, and test sets, these confusion matrices offer not only a way to visualize our data, but more specific information than precision and recall values offer.

We selected our hyperparameters based on empirical results and industry standards. For instance, choosing a 4 by 4 window for our first convolution window was a result of seeing a similar window size work in other academic results, and then fine tuning to meet our data-specific needs. We chose to use Adam optimization for a few reasons. Working with audio time-series data over 2 dimensions causes sparse gradient problems, similar to those often encountered in natural language or computer vision problems. We also felt our data was somewhat noisy and messy. Adam mitigates the sparse gradient

problem by maintaining a per-parameter learning rate, and mitigates the noise problem by basing updates on a weighted average of recent updates (momentum). With Adam our models trained more quickly and didn't plateau as early.

Table 1: Accuracy of predictions by model used.

	Train	Test
K-Nearest Neighbors	1.00	0.67
Convolutional Neural Network	0.99	0.92

Consult Table 1 for model accuracy with data processing – i.e. using the scaled mel-spectrograms as input.

The clearest trend identifiable here is the dramatic jump in performance after we moved from using raw audio to mel-spectrograms. We saw a substantial improvement in accuracy on all four of our models after converting our data to this form, which suggests that there is essentially no benefit to looking directly at raw amplitude data. While it is true that converting to mel-spectrograms takes a little bit of time, especially with our high number of training examples, this preprocessing step can be pre-computed and stored in a file format such as .npz for quick access across all models. Additionally, mel-spectrograms are essentially images, as in Figure 1, which provides a human-accessible way to visualize our data and to think about what our neural networks may be classifying on. In other words, there is essentially no downside to switching to mel-spectrograms.

We also see that all four of our models struggle with over-fitting. We spent most of our time trying to mitigate this issue on CNN. To do so, we introduced three methods. We played around with a combination of batch normalization, dropout layers, and L2 regularization. We found some difficulty in using this, as allowing the model to over-fit actually increased our accuracy. While we could bring the training accuracy and the test accuracy to within .05 of each other, this would result in poor model performance. Thus, we accepted our overfitting issue.

Looking more closely at our confusion matrix, we see that our CNN struggled most with the rock genre. It only managed to correctly classify 50% of rock audio as rock, labeling the others as mainly country or blues. Additionally, it incorrectly classified some country, as well as a small fraction of blues and reggae, as rock music. While it's not all that surprising that rock was a challenging genre – a qualitative inspection of rock mel-spectrograms implies that many rock music excerpts lack the easily visible beats that other genres such as hip-hop and disco possess, while our personal experience with rock music vis a vis the other genres tell us that rock is also missing distinctive traits such as high-register vocals (pop) or easily audible piano (classical or jazz).

Additionally, rock is a genre that both encapsulates many different styles (light rock, hard rock, progressive rock, indie rock, new wave, etc.) and heavily influences many other derivative genres. However, we were surprised that rock and country were so easily confused, as opposed to rock and metal, which would seem to rely on more similar instrumentation and tempo. Additionally, we note that correct classification of jazz was less accurate than most other categories. Our algorithm falsely classified some jazz music as classical, although never did the reverse of this. We hypothesize that

the more piano-heavy jazz tracks may have been too close to classical music in terms of both tempo and instrumentation, and may have been missing the saxophone or trumpet sounds and timbres associated with many other samples of jazz audio.

5. CONCLUSION

Across all models, using frequency based mel-spectrograms produced higher accuracy results. Whereas amplitude only provides information on intensity, or how “loud" a sound is, the frequency distribution over time provides information on the content of the sound. Additionally, mel-spectrograms are visual, and CNNs work better with pictures. CNN performed the best, as we expected. It took the longest time to train as well, but the increase in accuracy justifies the extra computation cost. However, we were surprised to see the similarity in accuracy between the KNN, SVM, and feed-forward neural network. In the future, we hope to experiment with other types of deep learning methods, given they performed the best. Given that this is time series data, some sort of RNN model may work well (GRU, LSTM, for example). We are also curious about generative aspects of this project, including some sort of genre conversion (in the same vein as generative adversarial networks which repaint photos in the style of Van Gogh, but specifically for music). Additionally, we suspect that we may have opportunities for transfer learning, for example in classifying music by artist or by decade.

6. REFERENCES

https://www.clairvoyant.ai/blog/music-genre-classification-using-cnn

https://towardsdatascience.com/extract-features-of-music-75a3f9bc265d

https://medium.com/bisa-ai/music-genre-classification-using-convolutional-neural-network-7109508ced47

https://towardsdatascience.com/music-genre-classification-with-python-c714d032f0d8

https://www.analyticsvidhya.com/blog/2022/03/music-genre-classification-project-using-machine-learning-techniques/