The current task at hand is a multi-label music genre tagging task, where the target is to tag a given music track with zero, one or more of the 67 genre labels. In other words, the output is a 67 element long vector, indicating the genre labels for that track. Though the ground-truth data is given as a value between 0 - 100 indicating the strength of the genre for that track. We tackle the task as a multi-label music classification task and hence we transform the labels to binary - 1 (strength > 50) or 0 (strength < 50).

For audio feature representations, we use mfcc and mel spectrograms. These are commonly used feature representations for music classification tasks. Both these feature representations are extracted on the mel scale that is related to the perceived frequency of a pure tone. We take the first 120 seconds of the music track and extract these representations. We divide the 120 second clip into 10 segments and extract the features for each of these segments. We propagate the ground-truth labels at the track level to segment level, assuming that the genre does not change across the track. For mfcc, we extract 13 vectors for the segment resulting in a 517x13 matrix for each of the 10 segments. Similarly, we extract a 517x128 feature representation for the mel spectrogram.

We construct 2 types of models for this classification task - Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM). Both the models are vanilla with an adam optimiser and a learning rate of 0.0001. We use a batch size of 32 and run the models for 100 epochs. Each node in the output layer uses the sigmoid activation. This will predict a probability of class membership for the label, a value between 0 and 1. Both the models use the binary cross-entropy loss function. These loss and the activation functions are specifically used for multi-label classification. In addition, we use hamming loss as a metric for evaluating the model on both the validation and the test set. Hamming loss measures the proportion of misclassified examples. We used this article as a reference to choose an evaluation metric. Please run the following python file to run the entire pipeline: *python dlpipeline.py.* In the file *config.py,* we can modify the different parameters related to the dataset, model, feature extraction, and training. Please find the plot of training history for the LSTM model with MFCC feature representation (Figure 1). In order to avoid overfitting, we use three strategies while training the model - BatchNormalisation, Dropout and Early stopping. From the figure, we can see that these strategies seem to be helping to avoid overfitting. We use 134 music tracks as our test set and crop out a 12 second segment and perform feature extraction and then pass it to the trained model. We obtain a hamming loss of 0.03, which means that 3% of the labels are misclassified in our test set.

In addition to these models, we could potentially investigate how pre-trained image classification models like MobileNet would work for our classification task. In this case, we will be using s transfer learning strategy by removing the top layers of the pre-trained model and adding additional dense layers depending on the number of outputs - 67 in our case with the same binary cross-entropy loss function and a sigmoid activation function for the final output layer.

The codebase is divided into different classes, each indicating one of the steps of the deep learning pipeline - DataLoader, Extractor, Preprocessor, and DLModel. We have an additional class - DLPipeline that runs all the steps of the pipeline in order - split the data, feature extraction, pre-processing, training, and evaluation. We use explicit typing in each of the methods for easy readability of the code. In addition, we documented the functionality of each class and method using docstrings. We used abstract classes for Extractor and DLModel, as it will enable us to extend these classes for new features and models.

Broadly speaking, we used the following publicly available python libraries - tensorflow.keras (for deep learning), librosa (for feature extraction), pandas (for data processing), json (for loading and saving dictionaries in python), and matplotlib for generating the plots. I created a conda environment with the required libraries (requirements.txt is attached to the repository) with Python 3.8. There were no specific difficulties faced while working on the task except that the feature extraction process was slow using librosa. One of the reasons for this could be the format of the file being mp3. Converting them to the raw wav format would have sped up the process. Since the models are not heavy and the dataset not very big, the training process could be run smoothly on a macbook with no gpu support.

Following are some of the aspects I would have liked to spend some more time on:

- Other feature representations like spectrogram and also investigate how we could use raw audio as input to the DL models
- I used only 120 seconds of a music track as the duration was different for all the tracks. We could utilise padding strategies to bring the music tracks to the same length.
- Use data augmentation strategies like pitch shifting, time stretching etc. to increase the size and variety in the training set.
- Test with traditional machine learning classification algorithms
- Build a regression based solution for the challenge using metrics like mae or mse.
- Building a rest api and dockerising the application would provide an easy access to the classification algorithm