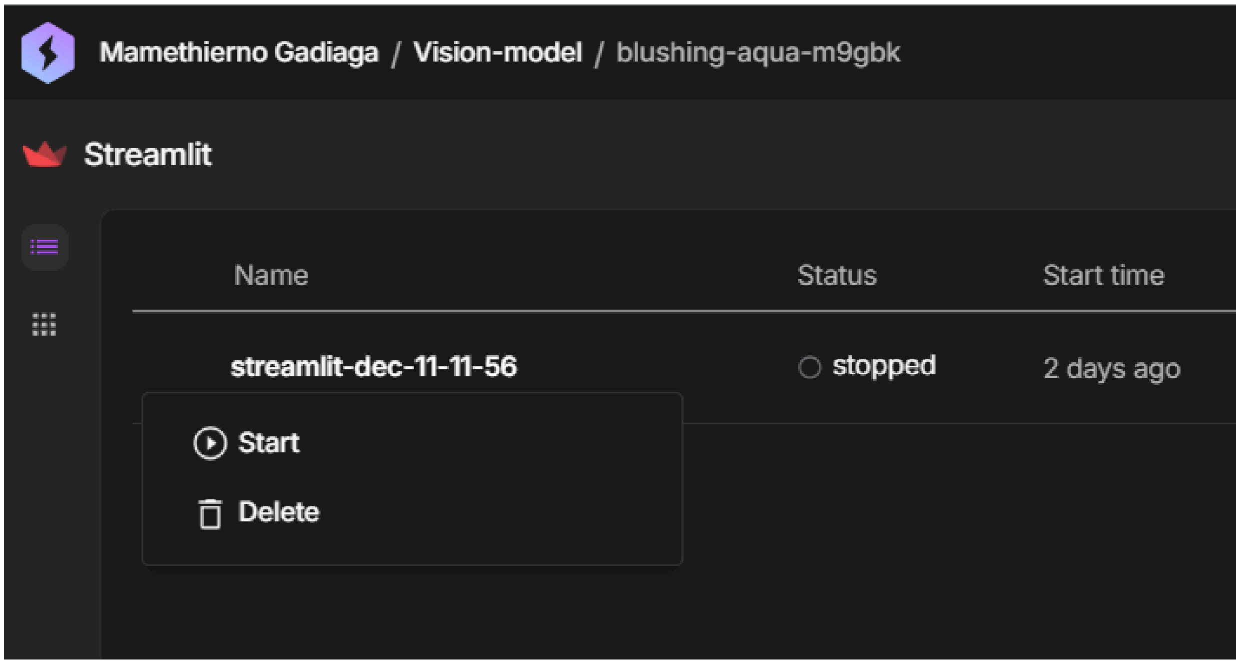
Group members:

- Mamethierno Gadiaga
- Richard Cenedella

Public GitHub Repository: https://github.com/thiernohgradiagram/csc_461_fp
Dev environment: https://lightning.ai/live-session/3e628c0a-b443-431a-b4a3-50fc05aab27c
Streamlit App: https://8501-01je6xynz1y1thrfff3cq8yzv8.cloudspaces.litng.ai

Notes about the app:
We gave you access to our development environment where you can access our code and start the app. You just have to open the link, it will ask you to create an account. After you create an account it should give you access to our development environment.
Once you have access to our development environment, you would be able to start the streamLit app by clicking on the streamLit icon on the right panel. You should now be able to start the app. Once you have started the app, you can share the link or QR code with anyone to access the app.



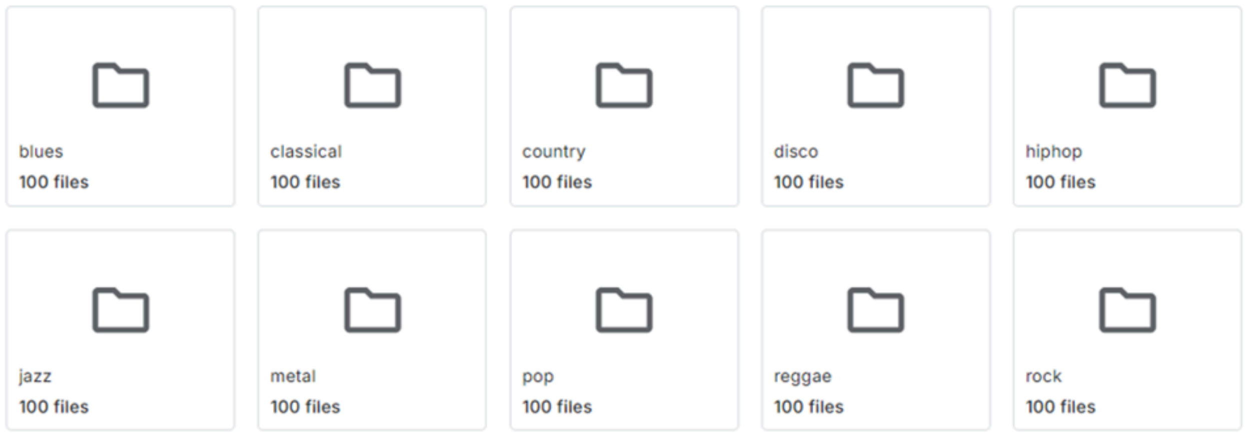
Project Proposal: Music Genre Classification Using Machine Learning

Main Idea

The primary objective of our project was to develop a machine learning model capable of classifying audio clips into one of ten distinct music genres. By analyzing the audio features of a given clip, the model would determine if it belonged to genres such as **blues, classical, country, disco, hiphop, jazz, metal, pop, reggae, or rock.** Our project aims to address the challenge of music genre recognition (MGR), which has applications in music recommendation systems, content categorization, and automated audio labeling. Classifying music genres accurately can be challenging due to the overlap in musical characteristics across different genres. Audio classification involves extracting relevant features from audio files (e.g., frequency, rhythm, and timbre) and training a machine learning model to recognize patterns associated with each genre. By accurately identifying these patterns, we hope to build a robust classifier for the given audio genres. We also wanted to make it easy for an end user to use the model which is why we built an easy to use website using streamlit. This is a python framework that is very similar to Gradio and allows us to easily implement our machine learning models into a frontend for an end user to use. This is the primary means of interacting with the model. We have developed models using the following Machine learning algorithms: **Multinomial Logistic Regression (MLR), Multilayer Perceptron Network (MLP) and finally a Convolutional Neural Network (CNN).** Our KNN, MLR, MLP model share the same feature extraction but our CNN mode uses a different feature extraction approach.

The dataset

For our project, we used the original, unprocessed GTZAN Dataset - A famous collection of 10 genres with 100 audio files each, all having a length of 30 seconds. The files were collected in 2000-2001 from a variety of sources including personal CDs, radio, microphone recordings, in order to represent a variety of recording conditions. The original, raw, unprocessed data can be downloaded from kaggle: https://www.kaggle.com/datasets/carlthome/gtzan-genre-collection



blues 100 files	classical 100 files	country 100 files	disco 100 files	hiphop 100 files
jazz 100 files	metal 100 files	pop 100 files	reggae 100 files	rock 100 files

Data preprocessing
After setting up the data into our pipeline using this notebook: csc_461_fp/notebooks/_01_data_setup.ipynb, we preprocessed each audio file using this notebook: csc_461_fp/notebooks/_02_data_preprocessing.ipynb

Why do we need to preprocess the audio files?
Preprocessing steps are generally necessary before feature extraction to ensure consistency, remove noise, and improve the quality of the features extracted.

1. Normalize Audio
- Why?
  - Normalization scales the amplitudes of the audio signal to a consistent range, which can:
    - Avoid biasing features toward audio files with higher or lower volume.
    - Ensure numerical stability during feature extraction (especially for RMS energy).
- How?
  - Normalizing each file to the range [-1,1] or to a consistent amplitude helps eliminate volume differences between clips.

2. Resample Audio
- Why?
  - Resampling to a common sample rate (e.g., 22.05 kHz) ensures:
    - Consistency in time and frequency features across the audio files.
    - Comparability of extracted features, since the sample rate affects the time and frequency resolution.
    - Reduced computational cost for higher sample rates (e.g., 44.1 kHz).
- How?
  - If the dataset includes audio clips at different sample rates, resampling ensures uniformity.
  - We don't need resampling because the sample rate for all audio files in our dataset is 22.05 kHz.

3. Trim Silence
- Why?
  - Trimming silence removes irrelevant portions of audio, which can:
    - Prevent silence from skewing features like zero-crossing rate, RMS energy, and MFCCs.
    - Focus the analysis on the actual audio content (e.g., music or speech).
- How?
  - Use a threshold-based method to detect and remove leading and trailing silence.

Feature extraction for our MLR and MLP models

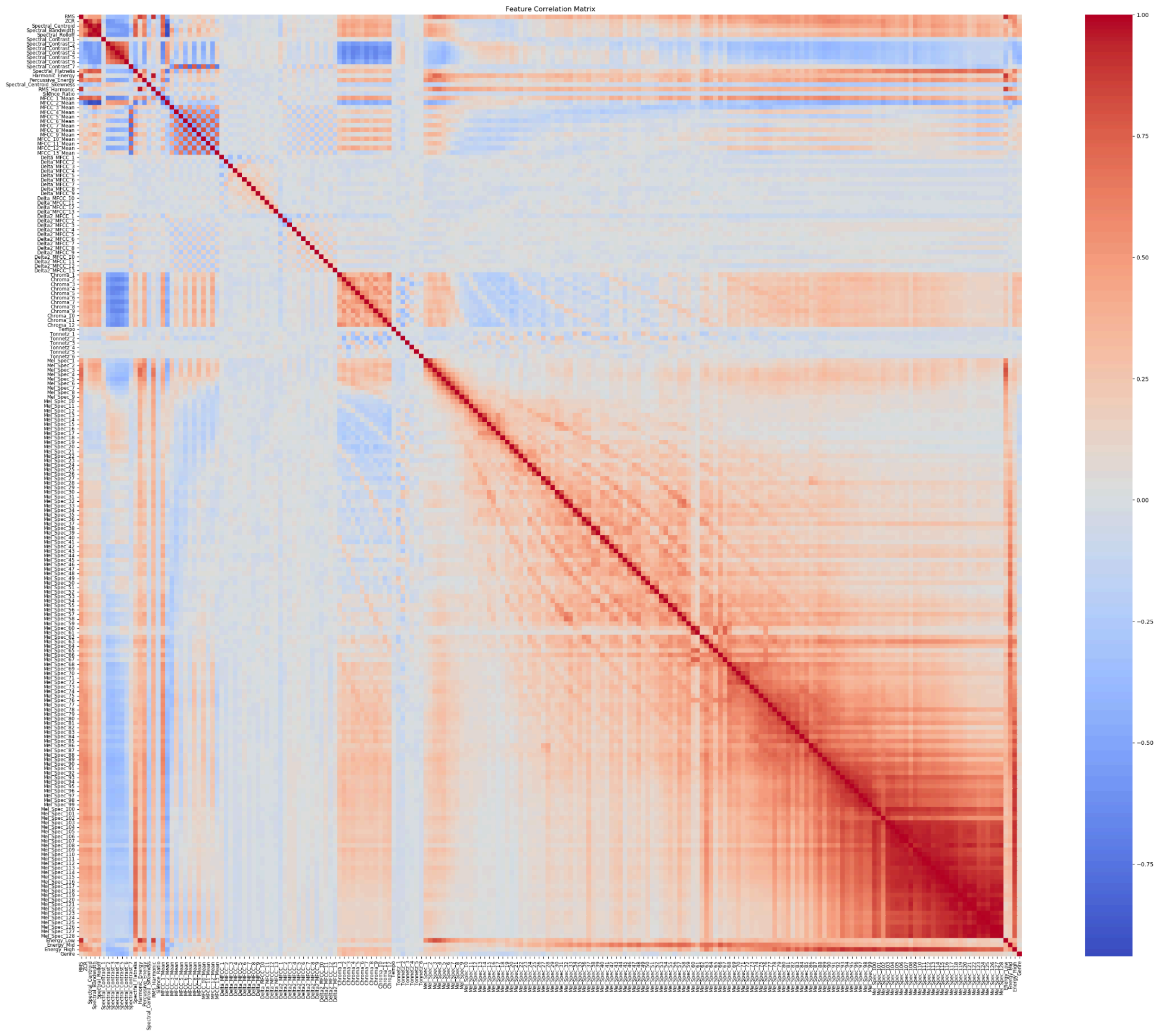
The features we extracted from each audio file can be found under the getColumnNames method from this module: csc_461_fp/features_extractor.py. We extracted different types of audio features from each audio files:
Time-Domain Features, Frequency-Domain Features, Harmonic and Percussive Energy, Spectral Statistics, RMS of Harmonics, Silence Ratio, Cepstral Features (MFCCs), Chroma Features, Tempo Features, Tonnetz Features, Mel Spectrogram Features

Many of the audio features from the above feature group are multi-dimensional. However, we wanted to experiment with an MLR and MLP which require an 1-d array of values as input [x1, x2, x3, x4, …, xn]. So we extracted features such that each of those multi-dimensional features were converted into smaller dimensions, often to 1 single value. That way of extracting the features for the MLR and MLP resulted in a significant loss of information, which was ultimately reflected on the performance of those models. After feature extraction, we ended up with a

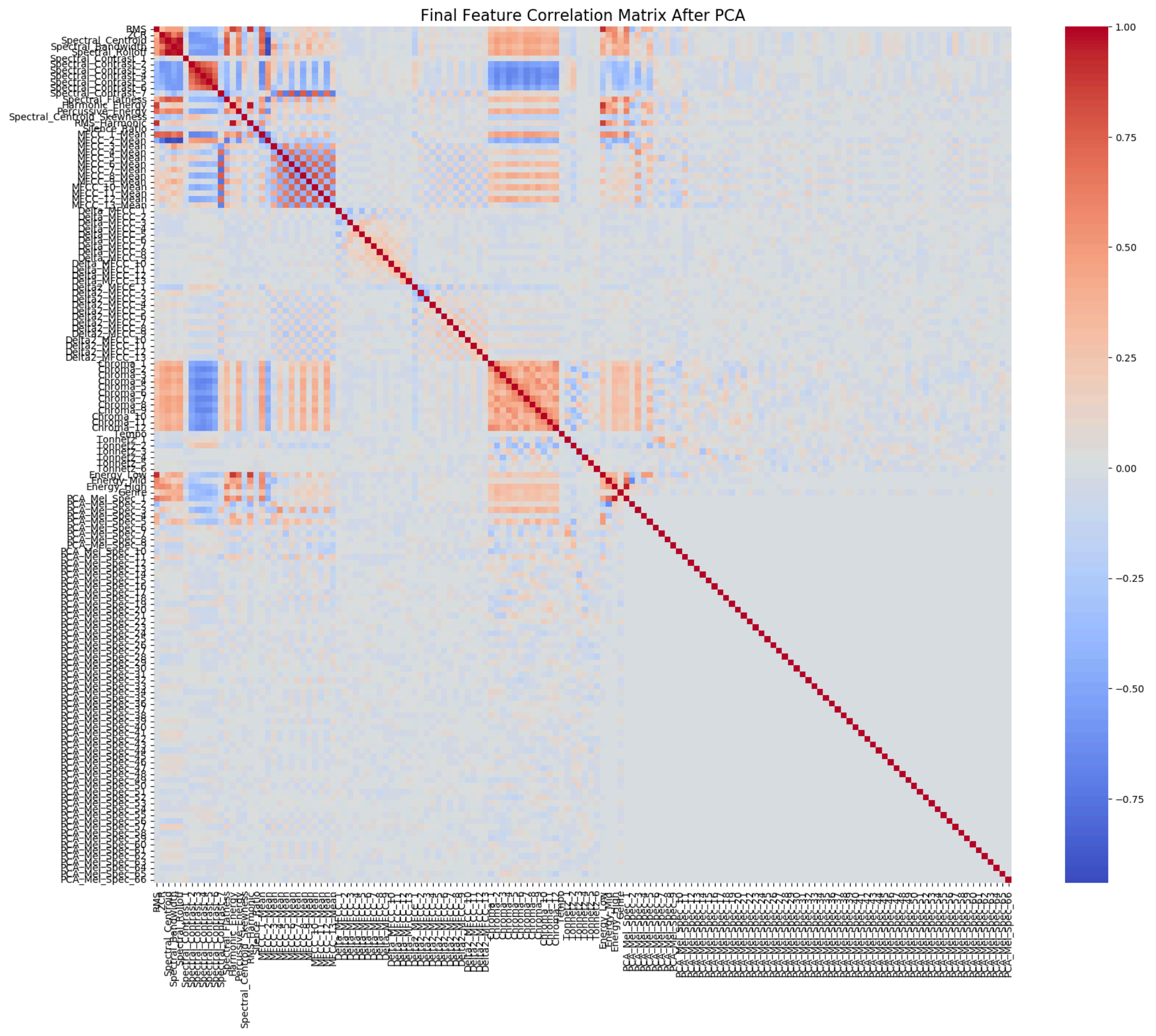
matrix of shape (1000, 207) where 1000 represents the number of rows (each audio) and 207 representing the number of features extracted for each audio. After feature extraction we need some Exploratory Data Analysis to investigate the features extracted. We found out that some of the features we extracted (the Mel spectrograms) are highly correlated as seen below.
So, we applied PCA to the mel spectrograms features to retain the components that hold 95% of the variance, which helped us reduce the high correlation. After feature extraction, we saved the extracted feature in an excel and a numpy file to avoid re-extracting the features for future runs.

Feature extraction notebook: csc_461_fp/notebooks/_03_features_extraction_cnn_demo.ipynb
EDA notebook: csc_461_fp/notebooks/_04_exploratory_data_analysis.ipynb
File containing the extracted features: csc_461_fp/_03_data_gtzan_features_labels_pca.xlsx

Correlation Matrix - Before PCA

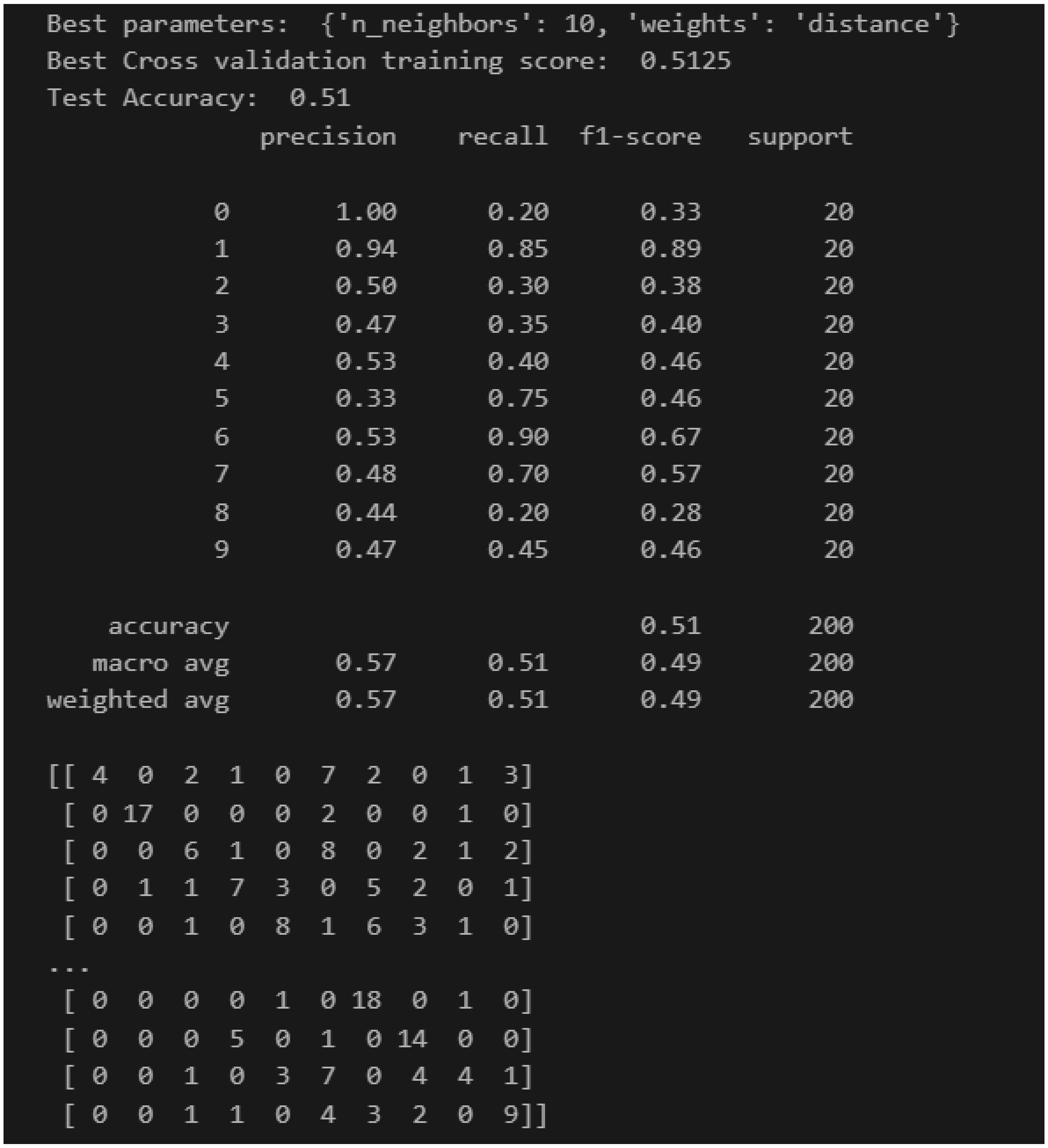


Correlation Matrix - After PCA



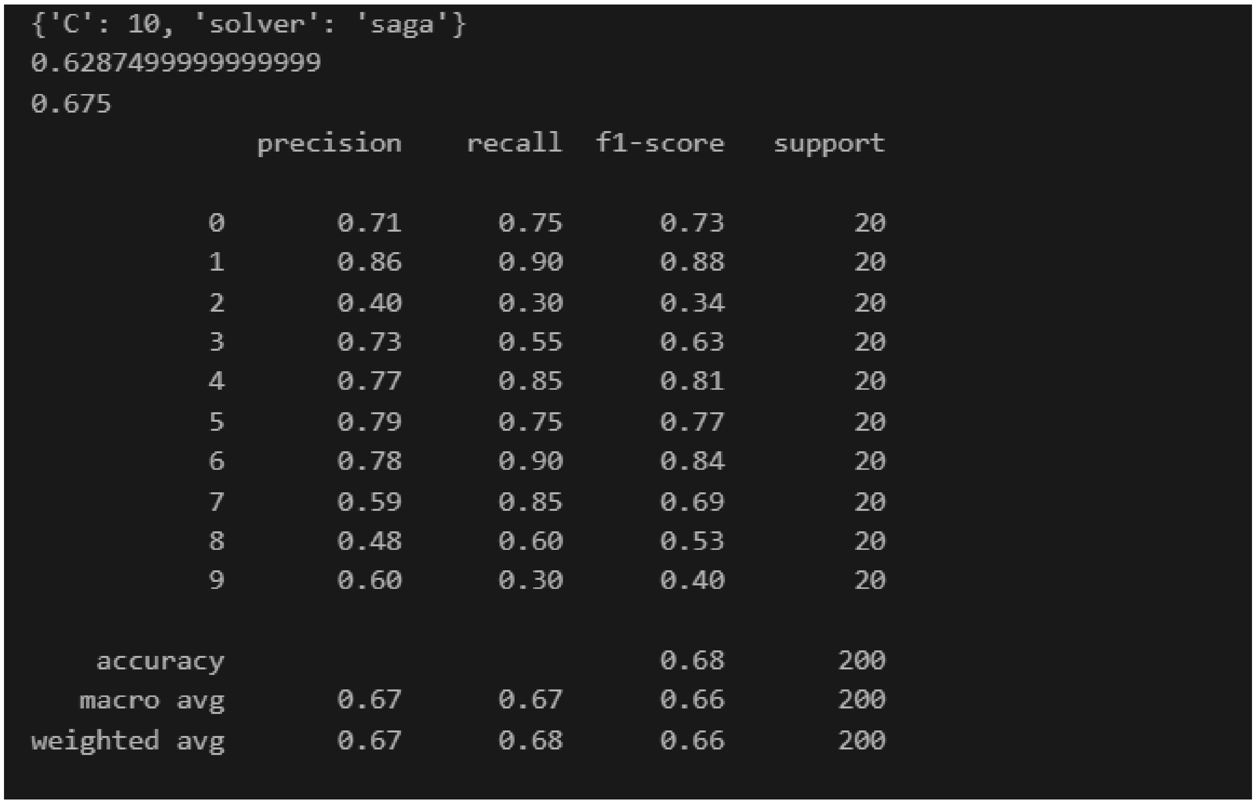
Experiments and Analysis for KNN and MLR
The notebook for those 2 models: csc_461_fp/notebooks/_05_knn_and_mlr.ipynb

We first started by implementing k-nn to see how well that could classify the genres.
We were able to achieve a Test accuracy of 51% as seen below



```
Best parameters:  {'n_neighbors': 10, 'weights': 'distance'}
Best Cross validation training score:  0.5125
Test Accuracy:  0.51
              precision    recall  f1-score   support

           0       1.00      0.20      0.33        20
           1       0.94      0.85      0.89        20
           2       0.50      0.30      0.38        20
           3       0.47      0.35      0.40        20
           4       0.53      0.40      0.46        20
           5       0.33      0.75      0.46        20
           6       0.53      0.90      0.67        20
           7       0.48      0.70      0.57        20
           8       0.44      0.20      0.28        20
           9       0.47      0.45      0.46        20

    accuracy                           0.51       200
   macro avg       0.57      0.51      0.49       200
weighted avg       0.57      0.51      0.49       200

[[ 4  0  2  1  0  7  2  0  1  3]
 [ 0 17  0  0  0  2  0  0  1  0]
 [ 0  0  6  1  0  8  0  2  1  2]
 [ 0  1  1  7  3  0  5  2  0  1]
 [ 0  0  1  0  8  1  6  3  1  0]
...
 [ 0  0  0  0  1  0 18  0  1  0]
 [ 0  0  0  5  0  1  0 14  0  0]
 [ 0  0  1  0  3  7  0  4  4  1]
 [ 0  0  1  1  0  4  3  2  0  9]]
```

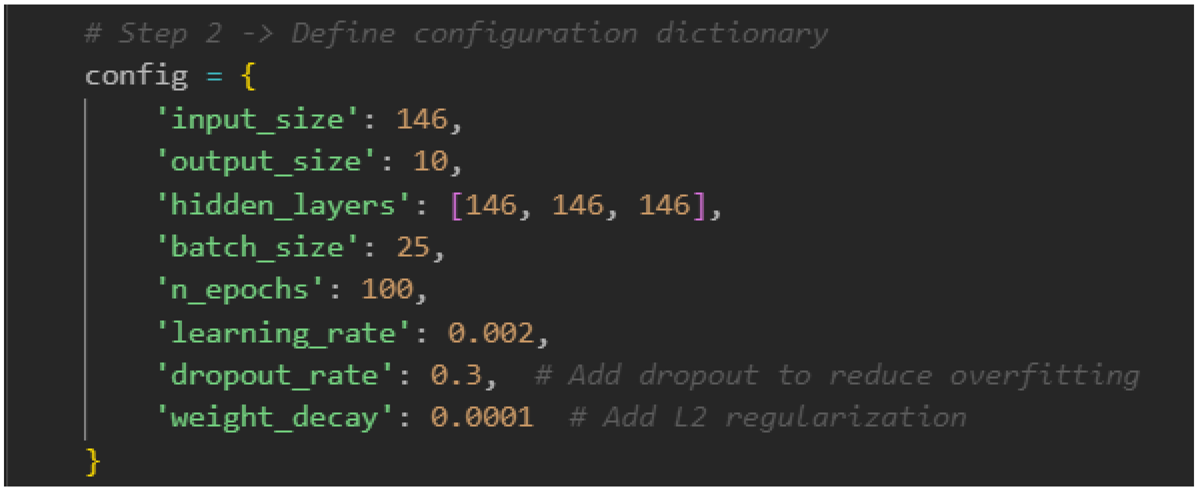
The next experiment was implementing a multinomial logistic regression model. We were able to achieve an accuracy of 67.5% on the test data once we chose the best parameters. These parameters turned out to be c=10 and solver='saga'. Just like k-nn, certain classes were able to be predicted with more reliability than others.



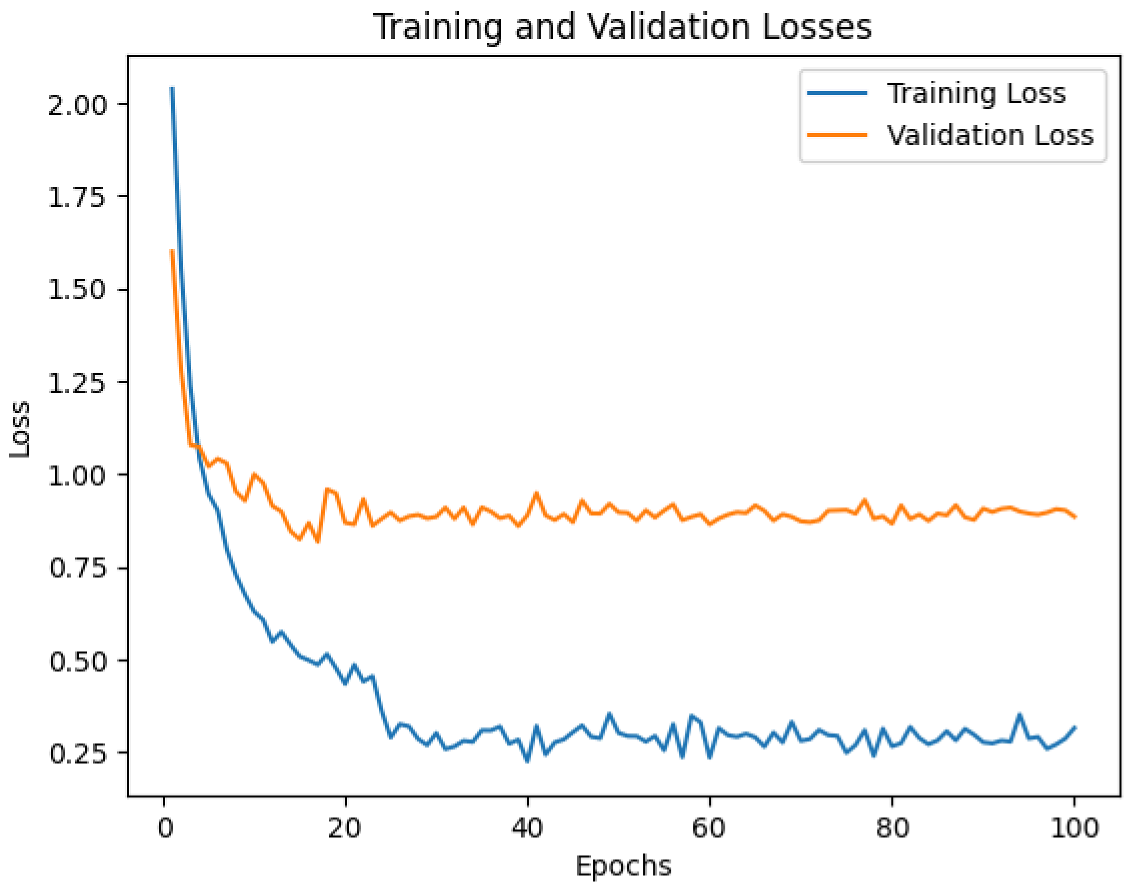
```
{'C': 10, 'solver': 'saga'}
0.6287499999999999
0.675
              precision    recall  f1-score   support

           0       0.71      0.75      0.73        20
           1       0.86      0.90      0.88        20
           2       0.40      0.30      0.34        20
           3       0.73      0.55      0.63        20
           4       0.77      0.85      0.81        20
           5       0.79      0.75      0.77        20
           6       0.78      0.90      0.84        20
           7       0.59      0.85      0.69        20
           8       0.48      0.60      0.53        20
           9       0.60      0.30      0.40        20

    accuracy                           0.68       200
   macro avg       0.67      0.67      0.66       200
weighted avg       0.67      0.68      0.66       200
```

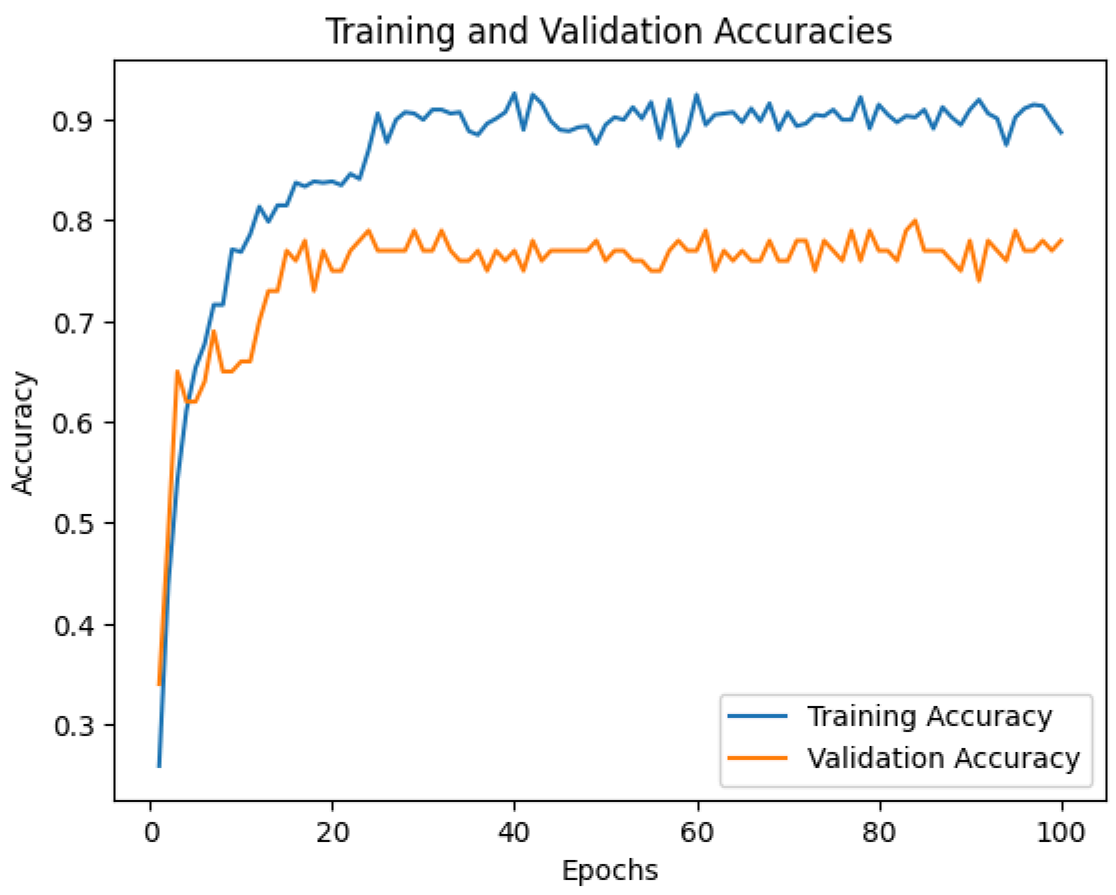
Experiments and Analysis for our MLP

The next model we implemented was an MLP network and tried a lot of different hyperparameters. The notebook is here: csc_461_fp/notebooks/_06_mlp_nn.ipynb.
Initially our training, validation, and test scores were hovering around 40% accurate. We tried changing the number of hidden layers, adjusted to different mini batch sizes, tried different learning rates, and adjusted the number of epochs. We also implemented batch normalization in the form of BatchNorm1d which had a very positive effect on improving our accuracy. Some other methods that we used to improve the model included implementing a scheduler called ReduceLROnPlateau to adjust the learning rate when it looked like the validation loss was not decreasing. We also included dropouts in the model to combat overfitting and used ReLU like before as the activation function. Unlike before we instead used AdamW for our optimizer. Another technique we used was early stopping to get better results.We were able to achieve a test accuracy of 72% using the following configuration



```
# Step 2 -> Define configuration dictionary
config = {
    'input_size': 146,
    'output_size': 10,
    'hidden_layers': [146, 146, 146],
    'batch_size': 25,
    'n_epochs': 100,
    'learning_rate': 0.002,
    'dropout_rate': 0.3,  # Add dropout to reduce overfitting
    'weight_decay': 0.0001  # Add L2 regularization
}
```





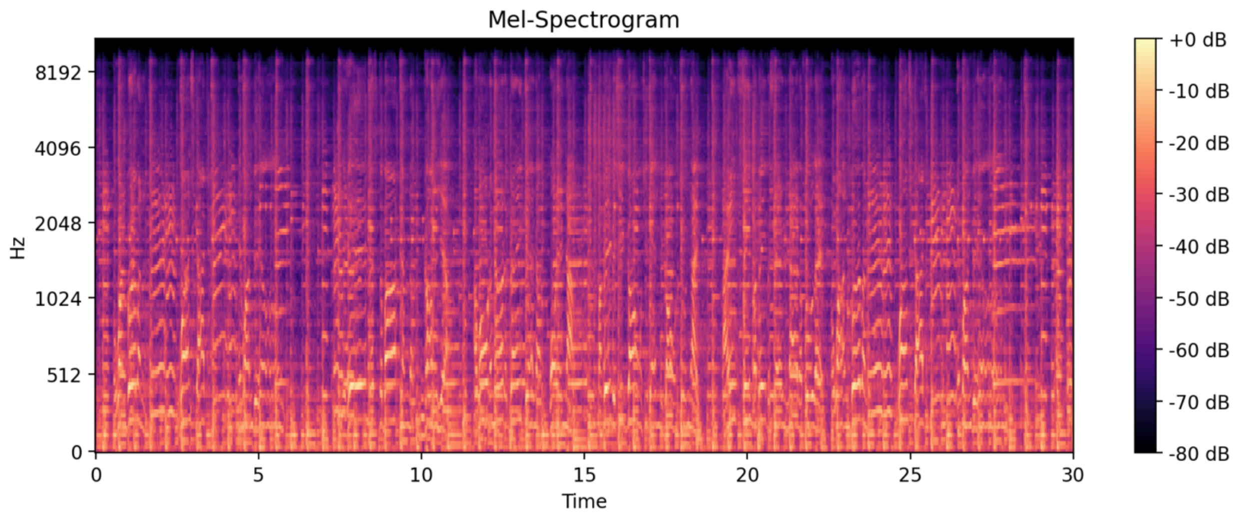
Experiments and Analysis of our CNN model

We were not satisfied with the 72% test accuracy that the MLP gave us, we wanted more, so we decided to implement a CNN model, it turned out well because we were able to achieve a test accuracy of 93%

Mel spectrogram

To train our model with a CNN we decided to extract a feature called Mel spectrogram from the audio file. A Mel spectrogram is a 2d array of shape (M, T) where M (y-axis) represents the number of Mel bands and T (x-axis) represents the time frame series.

Visual of a Mel Spectrogram of 1 audio file from our dataset



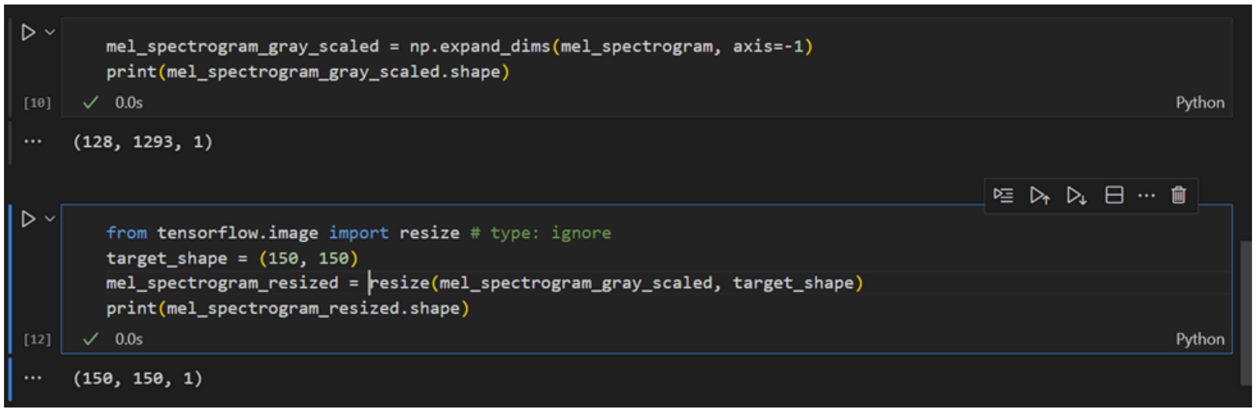
Why did we choose to extract Mel Spectrograms?

Human-Perceptual Frequency Scale:
The Mel scale is designed to align more closely with how human hearing perceives differences in pitch. Lower frequencies are represented with finer resolution, while higher frequencies are more coarsely spaced. This makes the features more perceptually meaningful, enabling the network to learn patterns more relevant to human perception of music and timbre.

Compatibility with CNN Architectures:
CNNs are highly effective at analyzing 2D data, such as images. By treating the Mel spectrogram as an "image" (time on one axis and frequency bands on the other), the CNN can leverage its spatial convolution operations to detect local patterns. These patterns might correspond to musical notes, chord progressions, drum patterns, or characteristic timbral fingerprints associated with certain genres.

Feature Extraction – extracting Mel Spectrogram from 1 Audio File

The time series of an audio signal is variable because it depends on the recording conditions. But we need a consistent shape for our CNN input layer, and we also need to treat the 2d Mel spectrogram as a grayscale image:



```
mel_spectrogram_gray_scaled = np.expand_dims(mel_spectrogram, axis=-1)
print(mel_spectrogram_gray_scaled.shape)
```

```
(128, 1293, 1)
```

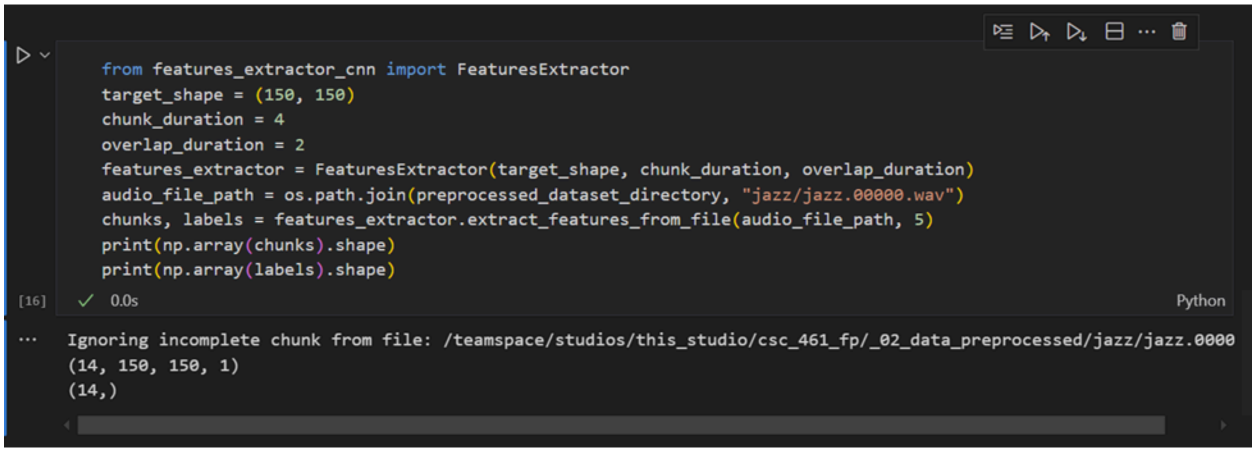
```
from tensorflow.image import resize # type: ignore
target_shape = (150, 150)
mel_spectrogram_resized = resize(mel_spectrogram_gray_scaled, target_shape)
print(mel_spectrogram_resized.shape)
```

```
(150, 150, 1)
```

Data Augmentation

We only have a 1000 audio files which is not enough to train our model with a CNN. So, we decided to divide each audio file into chunks of duration 4 seconds where the chunks would have an overlap duration of 2 seconds to prevent information loss.

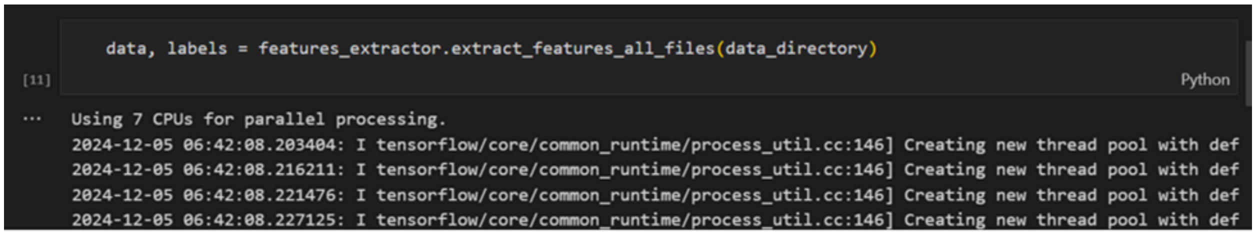
Here is how we chunk a single audio file: We have 14 chunks for the 1st jazz audio and each chunk can be seen as a grayscale image of shape (150, 150, 1)



```
from features_extractor_cnn import FeaturesExtractor
target_shape = (150, 150)
chunk_duration = 4
overlap_duration = 2
features_extractor = FeaturesExtractor(target_shape, chunk_duration, overlap_duration)
audio_file_path = os.path.join(preprocessed_dataset_directory, "jazz/jazz.00000.wav")
chunks, labels = features_extractor.extract_features_from_file(audio_file_path, 5)
print(np.array(chunks).shape)
print(np.array(labels).shape)
```

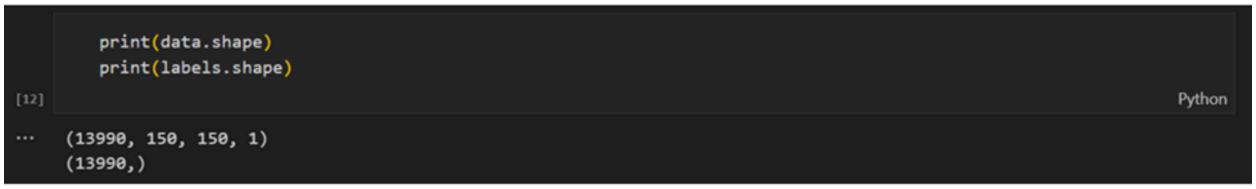
```
Ignoring incomplete chunk from file: /teamspace/studios/this_studio/csc_461_fp/_02_data_preprocessed/jazz/jazz.0000
(14, 150, 150, 1)
(14,)
```

After chunking all audio files, we end up with (13990, 150, 150, 1), each with their own label



```
[11]
data, labels = features_extractor.extract_features_all_files(data_directory)
```

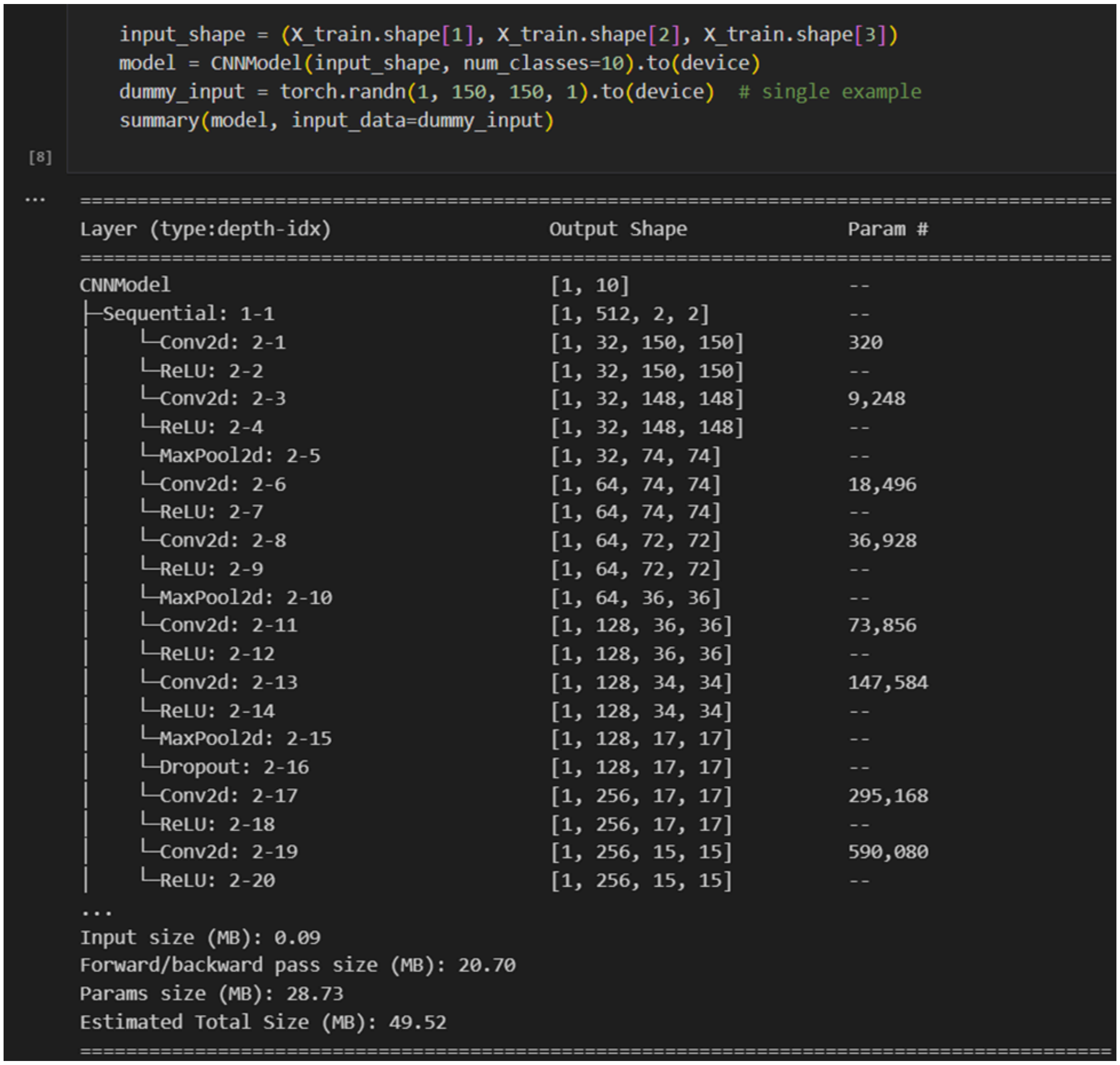
```
Using 7 CPUs for parallel processing.
2024-12-05 06:42:08.203404: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with def
2024-12-05 06:42:08.216211: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with def
2024-12-05 06:42:08.221476: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with def
2024-12-05 06:42:08.227125: I tensorflow/core/common_runtime/process_util.cc:146] Creating new thread pool with def
```



```
[12]
print(data.shape)
print(labels.shape)
```

```
(13990, 150, 150, 1)
(13990,)
```

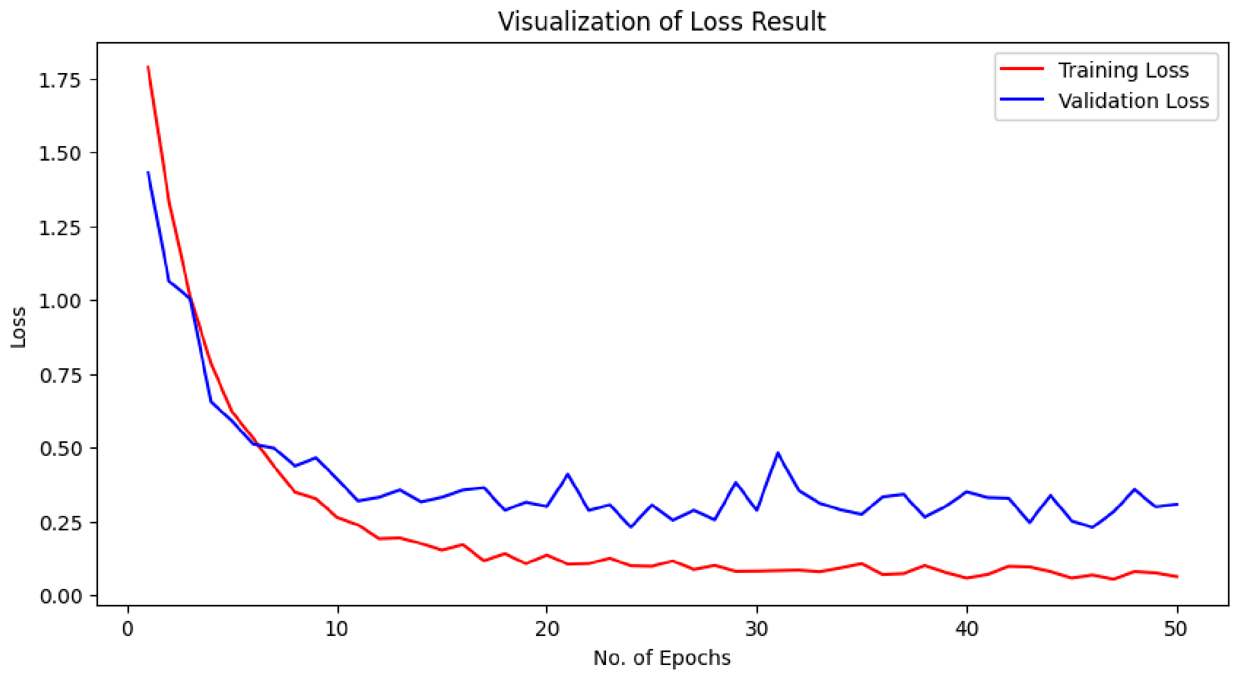
Here is our CNN architecture:



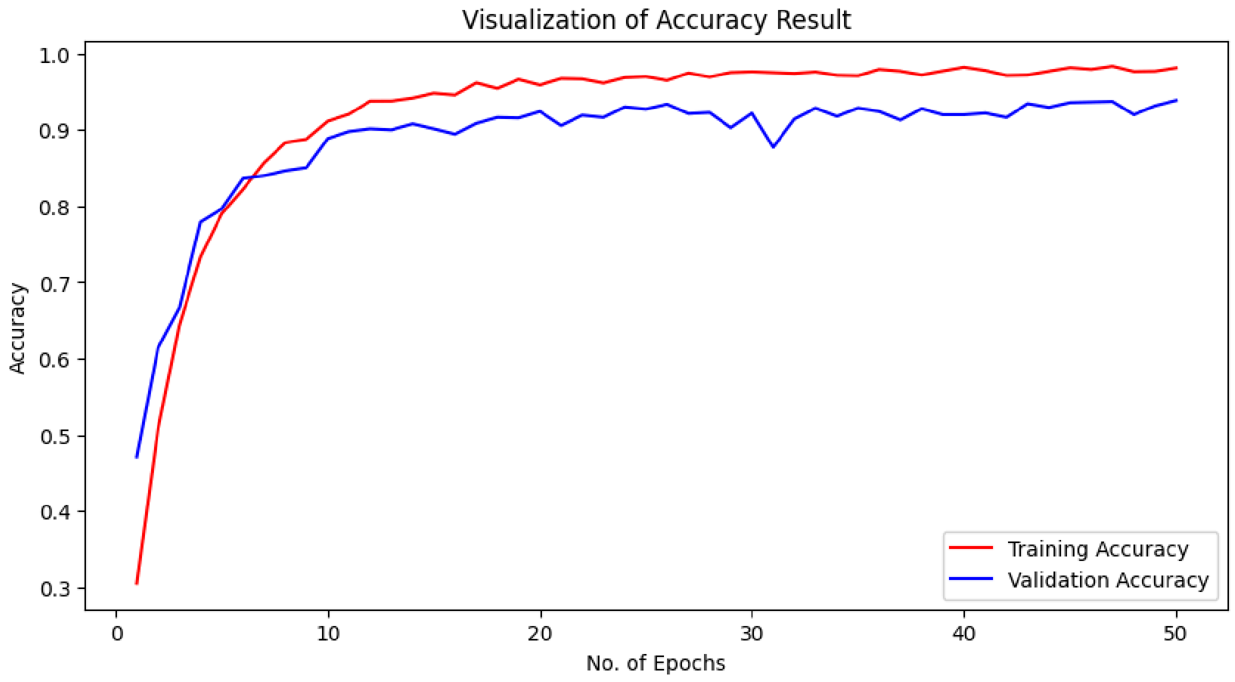
```
[8]
    input_shape = (X_train.shape[1], X_train.shape[2], X_train.shape[3])
    model = CNNModel(input_shape, num_classes=10).to(device)
    dummy_input = torch.randn(1, 150, 150, 1).to(device)  # single example
    summary(model, input_data=dummy_input)
```

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
CNNModel                                 [1, 10]                   --
├─Sequential: 1-1                        [1, 512, 2, 2]            --
│    └─Conv2d: 2-1                       [1, 32, 150, 150]         320
│    └─ReLU: 2-2                         [1, 32, 150, 150]         --
│    └─Conv2d: 2-3                       [1, 32, 148, 148]         9,248
│    └─ReLU: 2-4                         [1, 32, 148, 148]         --
│    └─MaxPool2d: 2-5                    [1, 32, 74, 74]           --
│    └─Conv2d: 2-6                       [1, 64, 74, 74]           18,496
│    └─ReLU: 2-7                         [1, 64, 74, 74]           --
│    └─Conv2d: 2-8                       [1, 64, 72, 72]           36,928
│    └─ReLU: 2-9                         [1, 64, 72, 72]           --
│    └─MaxPool2d: 2-10                   [1, 64, 36, 36]           --
│    └─Conv2d: 2-11                      [1, 128, 36, 36]          73,856
│    └─ReLU: 2-12                        [1, 128, 36, 36]          --
│    └─Conv2d: 2-13                      [1, 128, 34, 34]          147,584
│    └─ReLU: 2-14                        [1, 128, 34, 34]          --
│    └─MaxPool2d: 2-15                   [1, 128, 17, 17]          --
│    └─Dropout: 2-16                     [1, 128, 17, 17]          --
│    └─Conv2d: 2-17                      [1, 256, 17, 17]          295,168
│    └─ReLU: 2-18                        [1, 256, 17, 17]          --
│    └─Conv2d: 2-19                      [1, 256, 15, 15]          590,080
│    └─ReLU: 2-20                        [1, 256, 15, 15]          --
...
Input size (MB): 0.09
Forward/backward pass size (MB): 20.70
Params size (MB): 28.73
Estimated Total Size (MB): 49.52
==========================================================================================
```

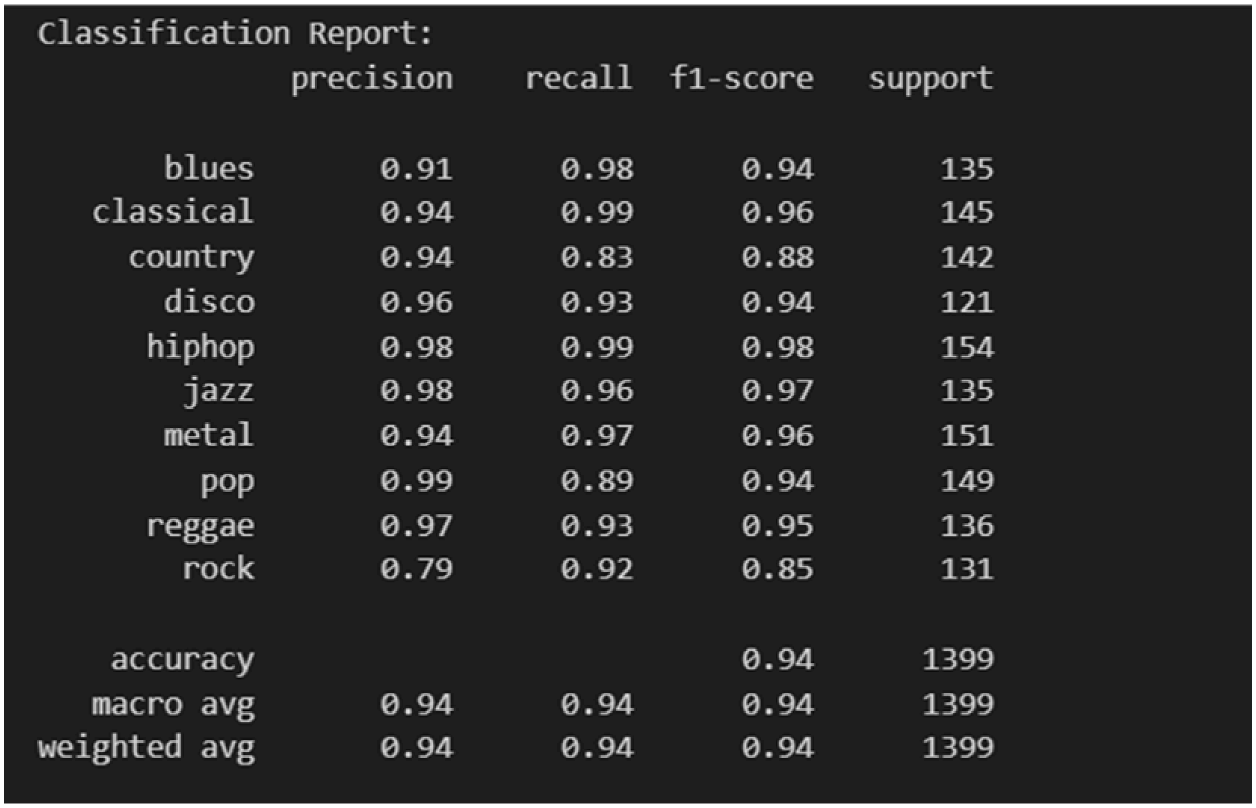
Results: Training Loss vs Validation Loss



Results: Training Accuracy vs Validation Accuracy

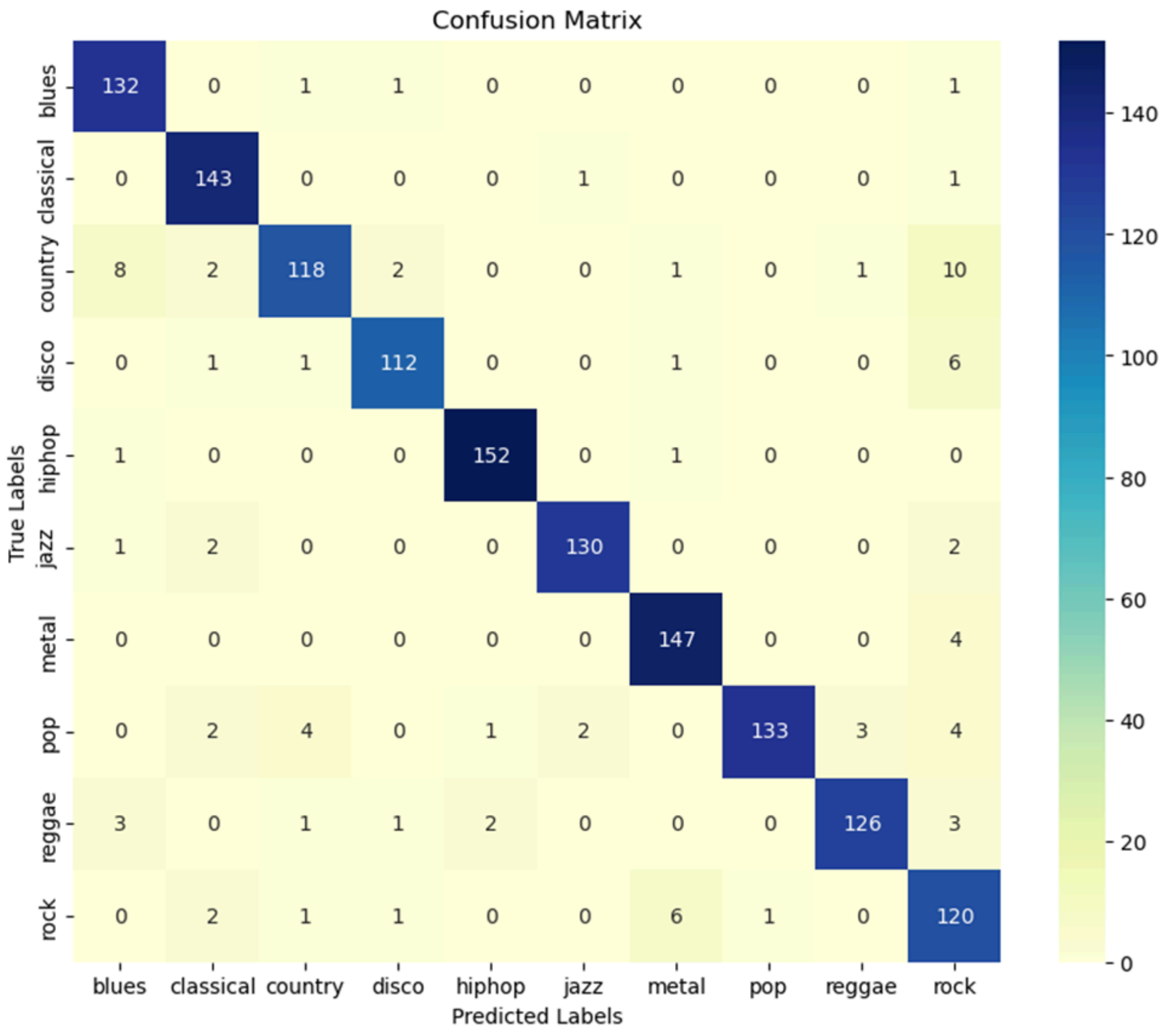


Classification Report:



```
Classification Report:
              precision    recall  f1-score   support

       blues       0.91      0.98      0.94       135
   classical       0.94      0.99      0.96       145
     country       0.94      0.83      0.88       142
       disco       0.96      0.93      0.94       121
      hiphop       0.98      0.99      0.98       154
        jazz       0.98      0.96      0.97       135
       metal       0.94      0.97      0.96       151
         pop       0.99      0.89      0.94       149
      reggae       0.97      0.93      0.95       136
        rock       0.79      0.92      0.85       131

    accuracy                           0.94      1399
   macro avg       0.94      0.94      0.94      1399
weighted avg       0.94      0.94      0.94      1399
```

Confusion Matrix:



	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	132	0	1	1	0	0	0	0	0	1
classical	0	143	0	0	0	1	0	0	0	1
country	8	2	118	2	0	0	1	0	1	10
disco	0	1	1	112	0	0	1	0	0	6
hiphop	1	0	0	0	152	0	1	0	0	0
jazz	1	2	0	0	0	130	0	0	0	2
metal	0	0	0	0	0	0	147	0	0	4
pop	0	2	4	0	1	2	0	133	3	4
reggae	3	0	1	1	2	0	0	0	126	3
rock	0	2	1	1	0	0	6	1	0	120

Insights:

Insight 1: The model performs very well on "blues" with minimal misclassifications.
Insight 2: Almost perfect classification for "classical”.
Insight 3: The model struggles with "country," with significant overlap with "blues" and "rock." This may be due to similarities in instrumentation or rhythm.
Insight 4: for disco, we have a good performance, but there are some overlaps with "rock."
Insight 5: for hip hop, excellent performance with very few misclassifications.
Insight 6: for jazz, we have a good performance but minor confusion with "country" and "rock.”
Insight 7: for metal, we have a strong performance but shows a slight overlap with "rock,"

possibly due to similar instrumentation.

Insight 8: The model struggles a bit with "pop," showing confusion with other genres like "country" and "rock."
Insight 9: for reggae, we have a good performance but overlaps slightly with "blues" and "rock"
Insight 10: The model struggles the most with "rock," showing confusion with "metal" and "pop." This is expected, as these genres often share similar features like distorted guitars and energetic rhythms.

Detailed Analysis of Blues, Classical, country, Distro, hiphop

1. Blues

Correct Predictions (Diagonal): 132 out of 135 (very high accuracy).
Misclassifications:
1 misclassified as "country."
1 misclassified as "disco"
1 misclassified as "rock."

2. Classical

Correct Predictions (Diagonal): 143 out of 145.
Misclassifications:
1 misclassified as "disco."
1 misclassified as "rock."

3. Country

Correct Predictions (Diagonal): 118 out of 142 (moderate performance).
Misclassifications:
8 misclassified as "blues."
10 misclassified as "rock."
2 misclassified as "classical."
2 misclassified as "disco."

4. Disco

Correct Predictions (Diagonal): 112 out of 121.
Misclassifications:
6 misclassified as "rock."
1 misclassified as "classical."
1 misclassified as "country."
1 misclassified as "pop."

5. Hiphop

Correct Predictions (Diagonal): 152 out of 154 (very high accuracy).
Misclassifications:
1 misclassified as "blues."
1 misclassified as "jazz."

Conclusions

This project was definitely a good learning experience for us as we learned a lot by trying different models and experimenting with different hyperparameters. We also learned a lot about how sound is represented numerically and how it could be used in a machine learning model. In conclusion, we found that a CNN model beats other models such as KNN, MLR, and MLP by far with a test accuracy of 93%.