Classifying Music Genre with Deep Learning

Quinn Olson

qolson@students.kennesaw.edu

4/28/2024

CS4277-01 - Deep Learning

Professor Zhao

Abstract

This project proposes the development of an AI-driven system for music genre classification, leveraging machine learning algorithms to address the inherent complexities and nuances in musical data. Despite significant advancements in music information retrieval, accurately classifying music into genres remains a challenging task due to genre ambiguity, the diversity of musical elements, and the subjective nature of genre labels. This system analyzes a diverse dataset across multiple genres to provide a scalable solution for music categorization, which holds potential benefits for music streaming services and digital libraries in organizing and recommending content.

Introduction

The digital age has revolutionized the way we access, consume, and interact with music. With millions of tracks available at an instant, the need for effective organization and classification of music has never been more apparent. Genre classification serves as a fundamental component of music metadata, facilitating efficient music retrieval, recommendation, and discovery. However, the task of accurately classifying music into genres is fraught with challenges, including the subjective definition of genres, the diversity of musical characteristics, and the overlap between different genres.

Related Work

Rong F. Audio Classification Method Based on Machine Learning. 2016 International Conference on Intelligent Transportation, Big Data & Smart City (ICITBS), Intelligent Transportation, Big Data & Smart City (ICITBS), 2016 International Conference on, ICITBS. December 2016:81-84. doi:10.1109/ICITBS.2016.98

In this paper, methods of extracting audio features are discussed. A hierarchical structure for audio data is proposed: frame, clip, shot, and high level semantic unity. Extracted features include short time energy, zero crossing rate, and Mel-Frequency cepstral coefficients. They use support vector machines with a gaussian kernel for classification, where training samples are mapped in a feature space. With this method, they achieved around 87% classification accuracy. It is important to note this accuracy is not necessarily on genre classification, but on various sounds.

Purwins, Hendrik and Li, Bo and Virtanen, Tuomas and Schluter, Jan and Chang, Shuo-Yiin and Sainath, Tara. Deep Learning for Audio Signal Processing. Institute of Electrical and Electronics Engineers (IEEE). 2019. http://dx.doi.org/10.1109/JSTSP.2019.2908700

This paper provides a comprehensive review of the application of deep learning techniques in audio signal processing, covering a range of topics including speech, music, and environmental sound processing. Dominant feature representations like log-mel spectra and raw waveform, and models including CNNs, LSTMs, and audio-specific neural networks are discussed. It also highlights applications in audio recognition, synthesis, and transformation, such as automatic speech recognition, music information retrieval, and source separation. Limitations mentioned include the need for large labeled datasets, computational complexity, and challenges in interpretability and adaptability of the models. The paper suggests that future research could address these limitations and explore the potential for cross-fertilization between different audio processing areas.

Methodology

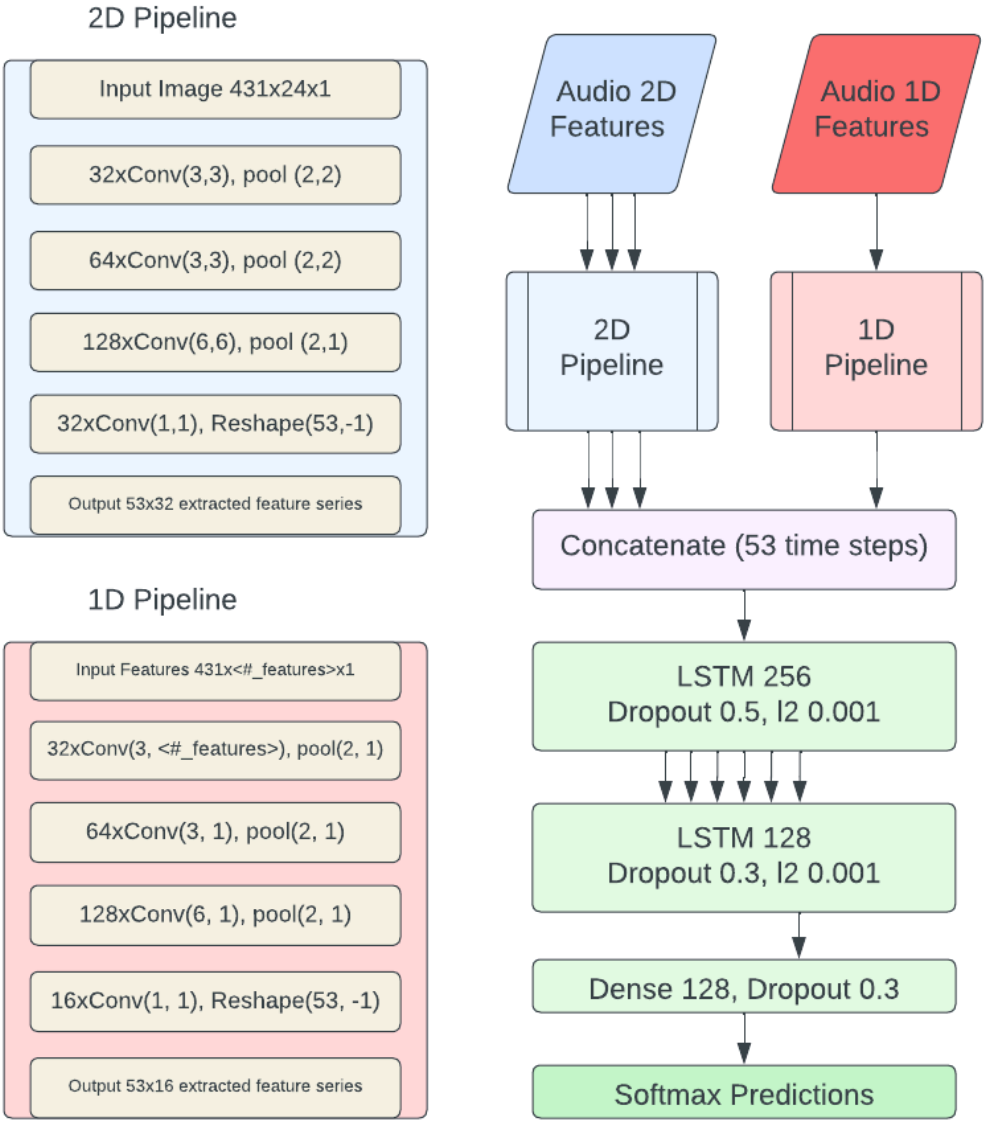
The crux of the proposed model lies in the synergistic use of convolutional neural networks (CNNs) and Long Short-Term Memory networks (LSTMs) to process and analyze audio features. In my design, 2-D features, which embody both feature and temporal dimensions, are conceptualized as images. These 'audio images' undergo a series of convolutions aimed at pattern extraction within the time-feature space. A critical design choice was made to process each type of audio "image" separately through identical convolutional pipelines, thereby maintaining their inherent temporal structure while avoiding any erroneous cross-correlation between disparate feature spaces.

Parallel to this, 1-D features are aggregated into a composite image, distinctly processed to obviate spatial correlation, ensuring that convolutions are exclusively temporal. The initial convolution layer compresses the feature dimension to unity, subsequently reshaping the output to position the depth dimension as the new feature space. This process is meticulously tuned to output a consistent number of frames, facilitating the merging of features into a cohesive temporal series.

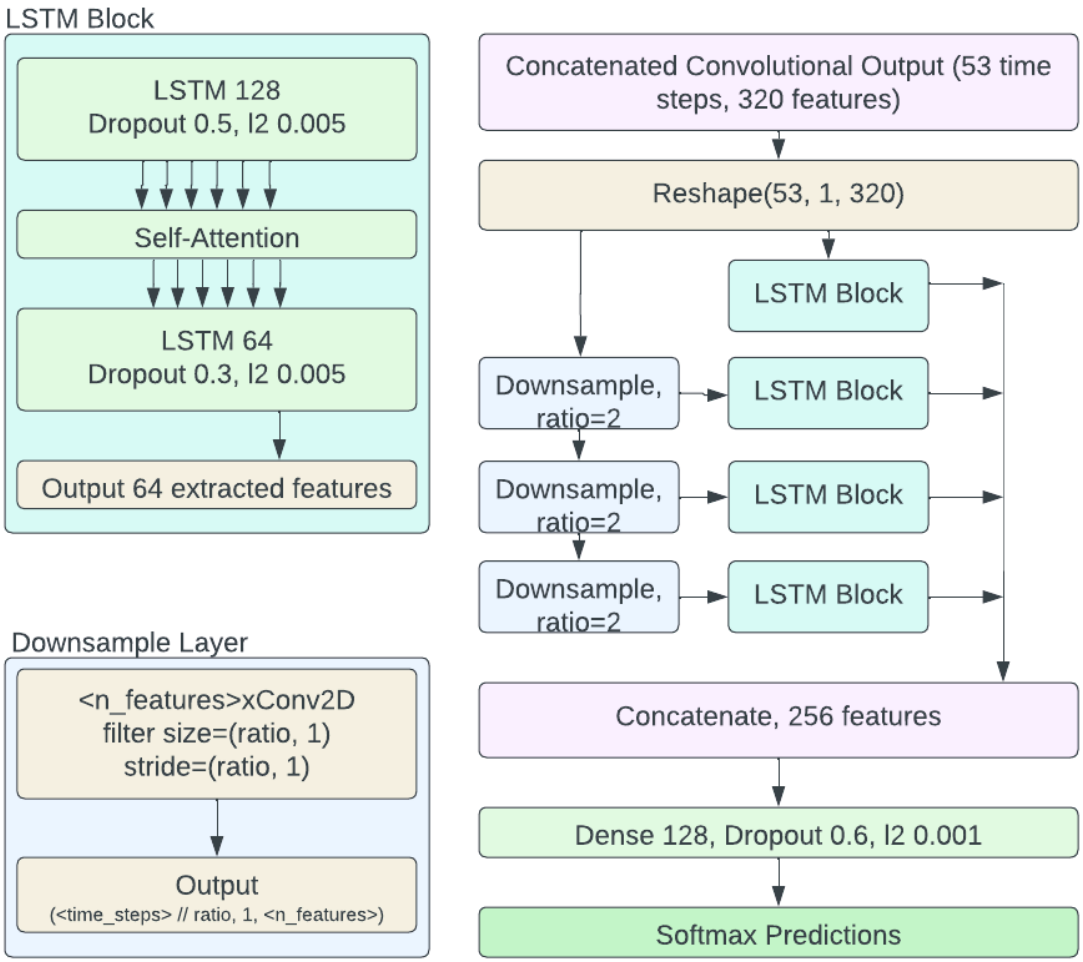
The resultant unified feature set, formatted as (time_frames, features), is then channeled through dual LSTM layers, designed to capture and model temporal dependencies and dynamics within the music sequences. The LSTM outputs are then consolidated by a dense layer, culminating in a softmax layer that probabilistically classifies the genres.

Throughout this pipeline, care has been taken to ensure temporal consistency across different feature representations. This consistency allows the LSTMs to effectively integrate information over time, providing a comprehensive understanding of each music clip's genre-defining characteristics. The combination of CNNs for feature extraction and LSTMs for sequence modeling encapsulates the complexity of musical patterns, laying the foundation for robust genre classification.

Model 1 Architecture



In my second iteration of the model, I retained the core architecture of the initial convolutional pipelines while enhancing their complexity to capture more nuanced features from the audio data. The primary modification in this iteration was the introduction of a multi-scale processing strategy in the temporal analysis stage. The down-sampling mechanism was implemented using convolutional layers and performed incrementally to generate multiple representations of the data, each at a different time scale. Each representation is then fed into its own LSTM block, consisting of two LSTM layers with attention in between them. The outputs from the multiple LSTM blocks are then concatenated to form a comprehensive feature set that integrates information across all considered time scales.

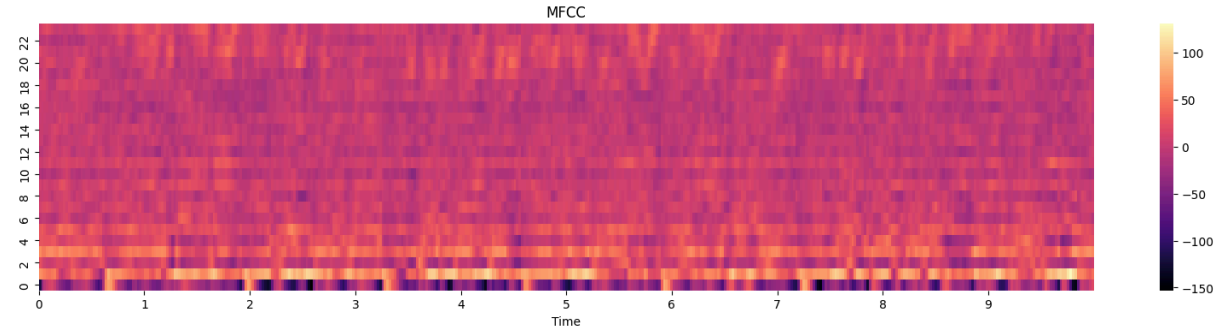
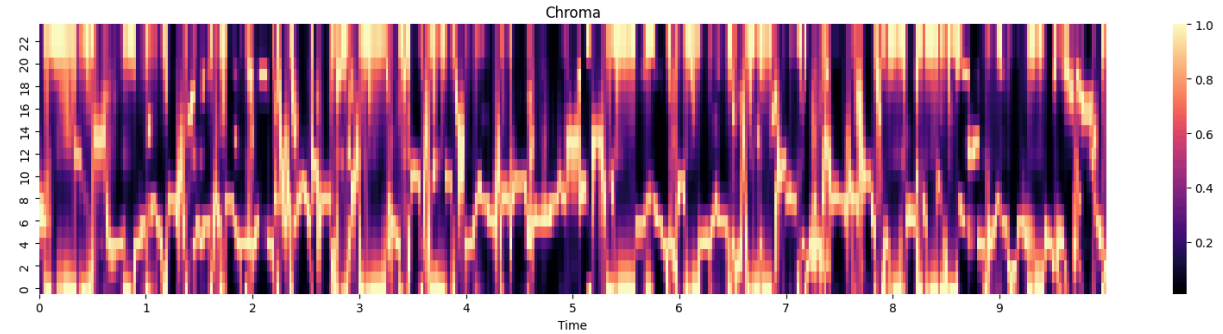


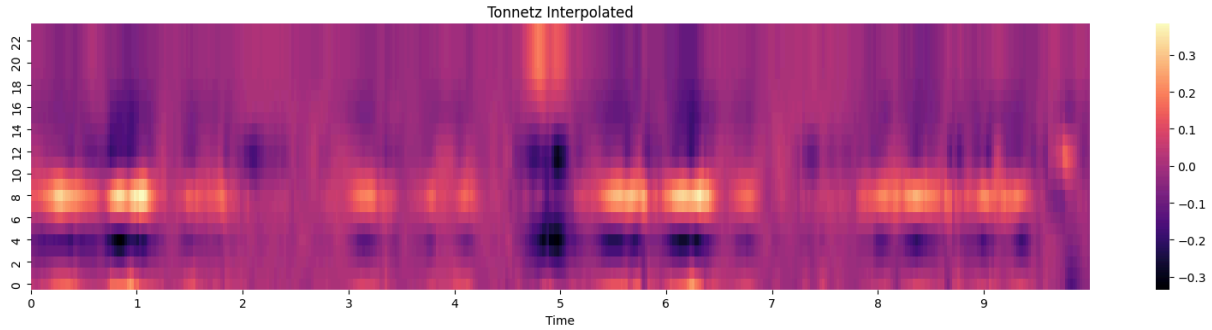
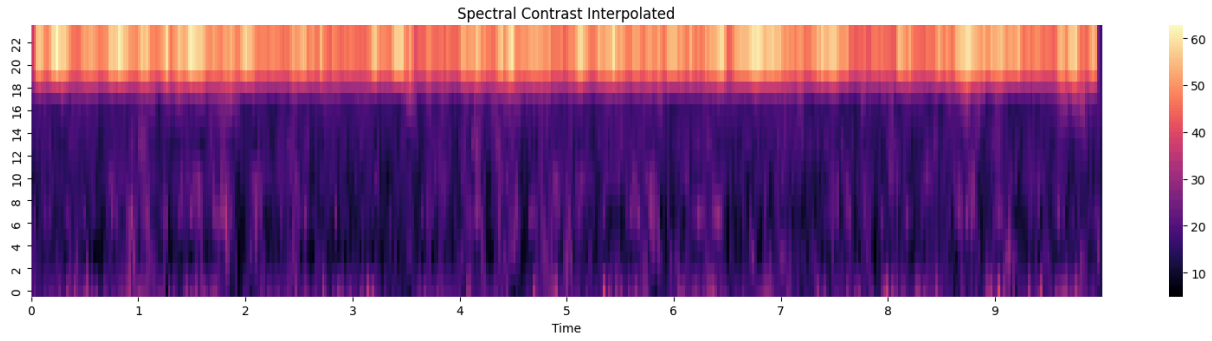
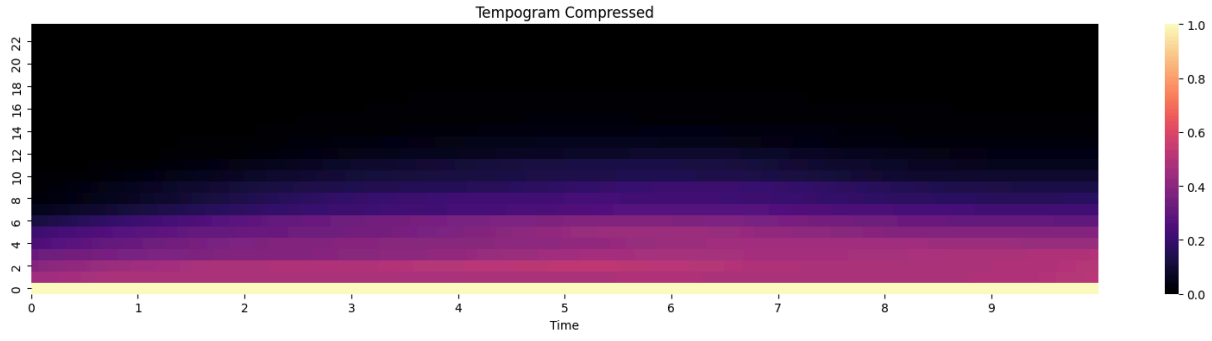
Experimental Setup

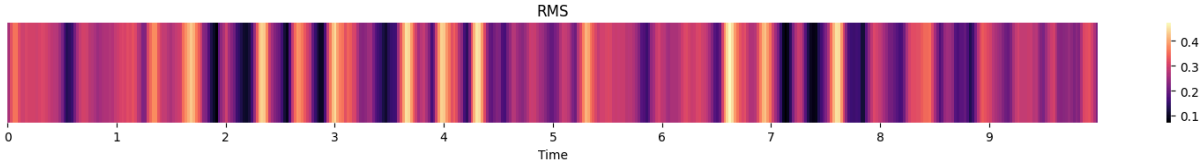
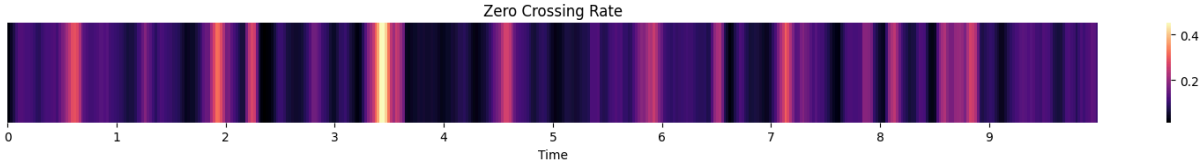
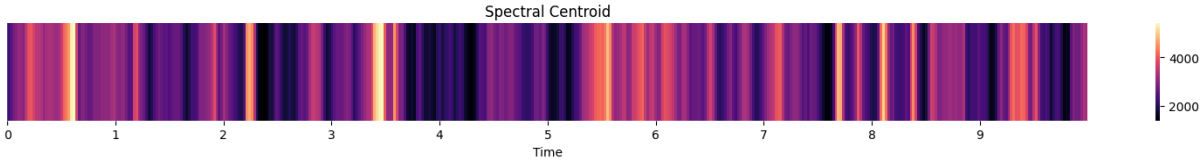
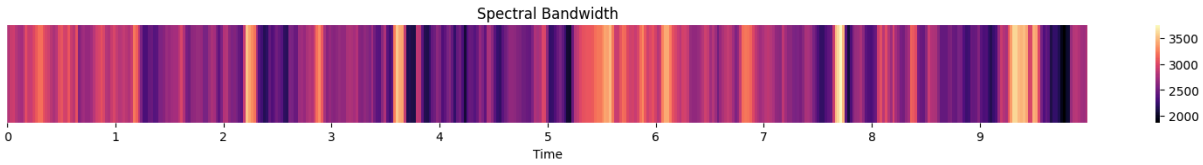
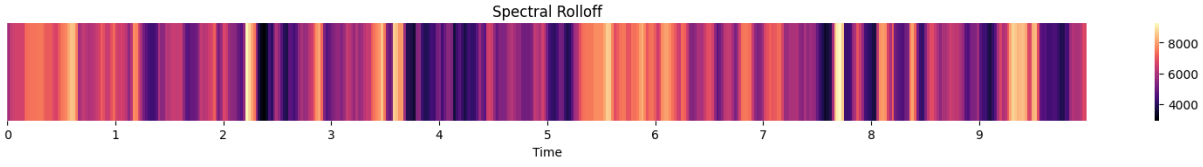
Dataset

My dataset is custom-built and consists of 12000 ten-second music clips scraped from internet sources, 1000 for each of twelve genres. Each clip is standardized to a ten-second duration to ensure uniformity. From these clips various audio features are extracted on a frame-by-frame basis, where each feature extracted results in either a 1-D or 2-D feature, either a single value or a vector over a period of 431 frames. All 2-D features are transformed via interpolation to match a consistent feature vector size of 24. In the final dataset, all features are minmax normalized feature-by-feature dataset-wide.

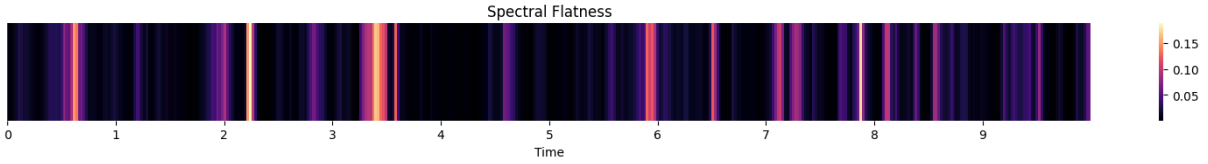
The current extracted features are as follows:

Feature	**Size**	**Description**
Mel-Frequency Cepstral Coefficients (MFCC)	(431, 24)	Timbre characteristics, audio aspects based on human perception.
		
Chroma Features	(431, 24)	Energy in each pitch class (C, C#, D, etc).
		

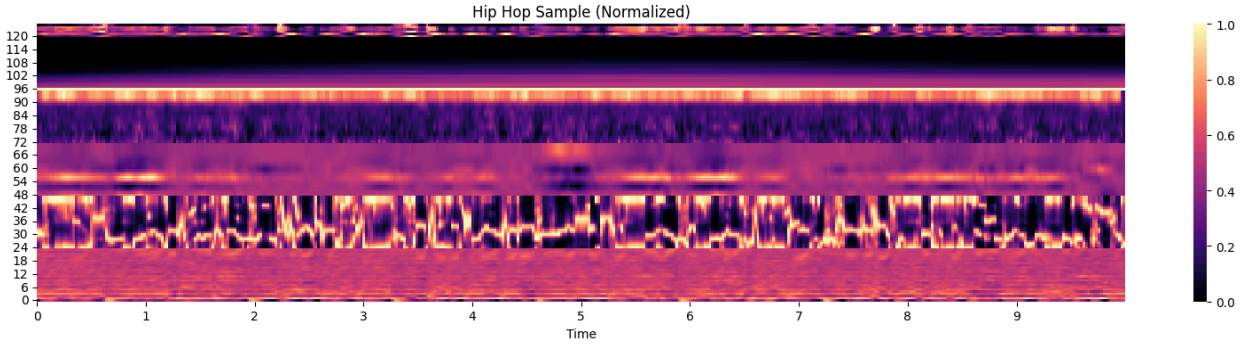
Tonnetz (Tonal Centroid Features)	(431, 24)	Tonal space. Captures harmonic relations and chord progressions.
		
Spectral Contrast	(431, 24)	Measures difference in amplitude between peaks and valleys in the sound spectrum across frequency bands. Highlights contrast between harmonic content and noise.
		
Tempogram	(431, 24)	A time-tempo representation.
		

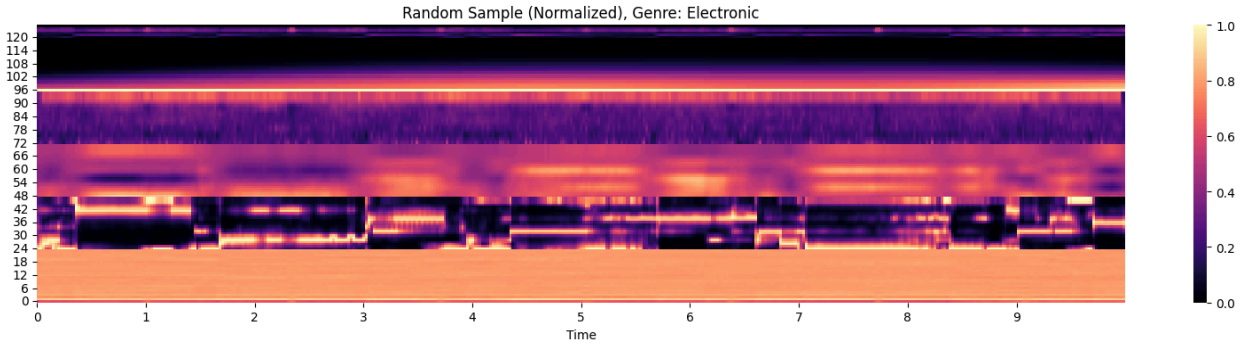
Root Mean Square Energy	(431, 1)	The loudness or intensity of sound.
		
Zero Crossing Rate	(431, 1)	The rate the audio signal changes sign over the audio frame. Can aid in differentiating percussive vs harmonic sounds.
		
Spectral Centroid	(431, 1)	Spectrum center of energy distribution. Higher values indicate higher frequencies.
		
Spectral Bandwidth	(431, 1)	The spread of frequencies around the centroid
		
Spectral Rolloff	(431, 1)	Frequency below which the majority of energy is contained. Estimates the boundary between noise and content
		

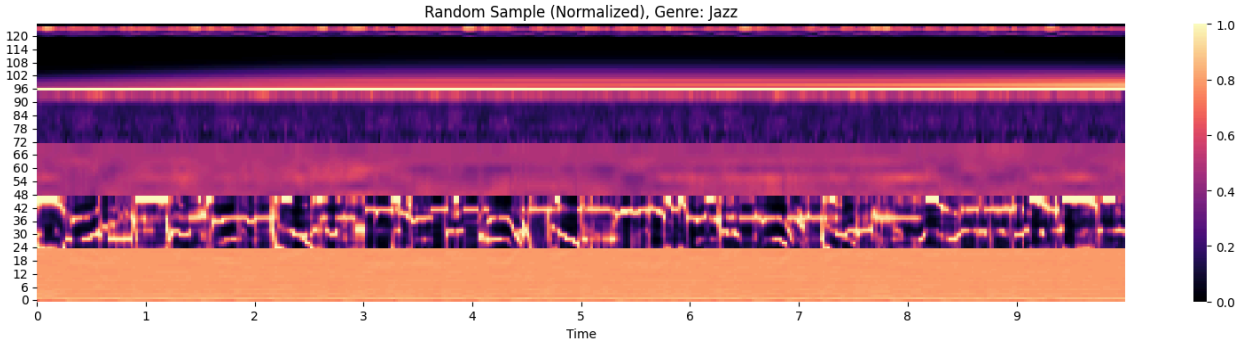
Spectral Flatness	(431, 1)	How evenly distributed frequencies are. Measures how noise-like sound is, as opposed to tonal-like.



Fully processed samples:







Evaluation

The model is evaluated by validation accuracy, confusion matrices, and by precision, recall, and f1-score. I implemented a custom confusion matrix that generates a distributed score for each genre by squaring and normalizing the softmax probabilities, intensifying the model's confidence levels and providing a richer picture of its predictive tendencies. Hierarchical clustering then arranges genres in a manner reflecting the model's perception of similarity, creating a genre-spectrum of sorts and offering deeper insights into its genre differentiation patterns.

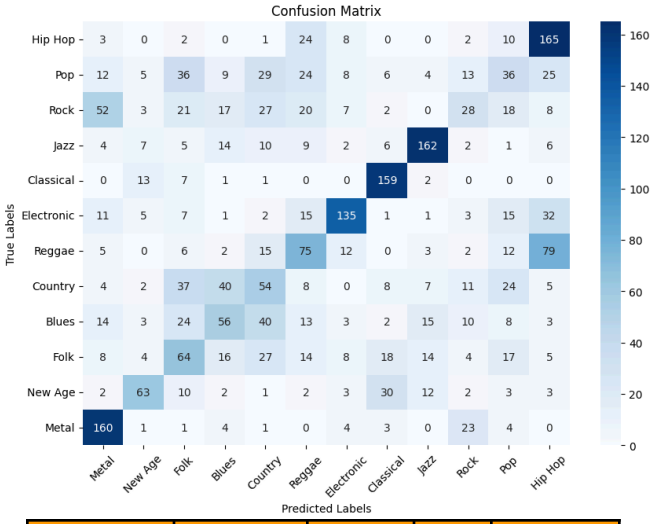
Implementation

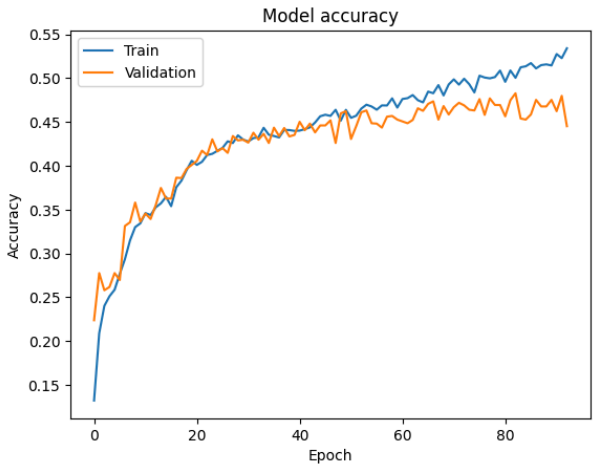
The model was constructed using the TensorFlow and Keras libraries and was trained on an RTX 3070 GPU within a WSL Ubuntu environment. This implementation is designed to be input-flexible for ease of iteration on which features I include in training. Attempts to train on all processed features results in resource issues that severely impede training time.

Results

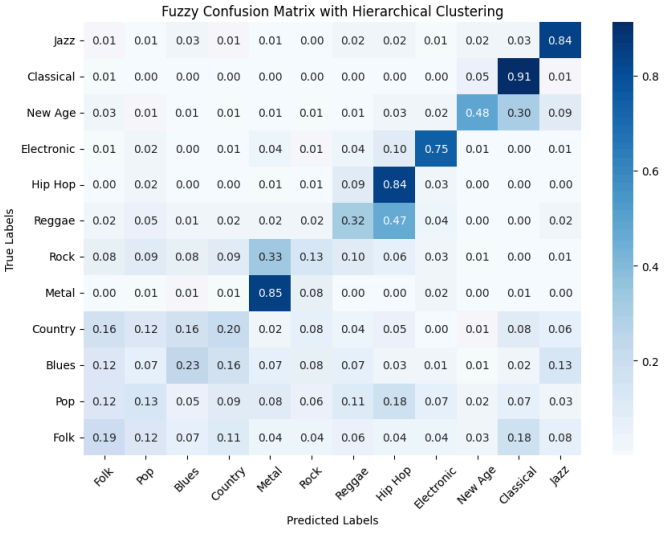
Model 1

The features used in this model are mfcc's, chromas, spec_contrast, tonnetz, zcr, rms, spec centroid, spec bandwidth, spec rolloff, and spec flatness. Stopped early after epoch 93, batch size of 200, and 16-bit precision. Training took roughly 30 minutes. 1.5 million parameters.



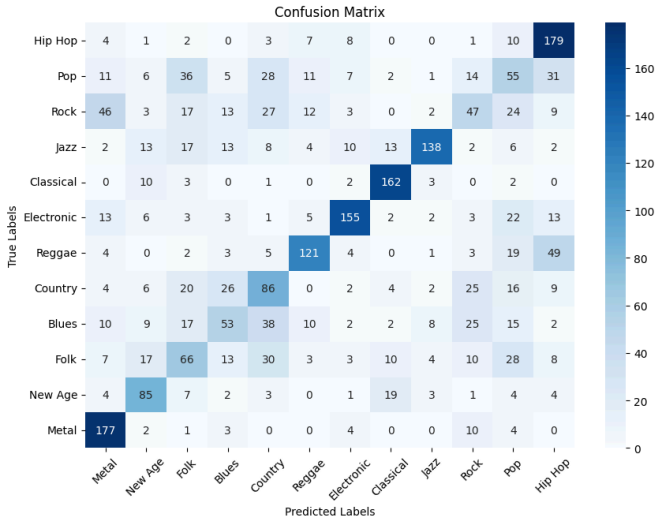


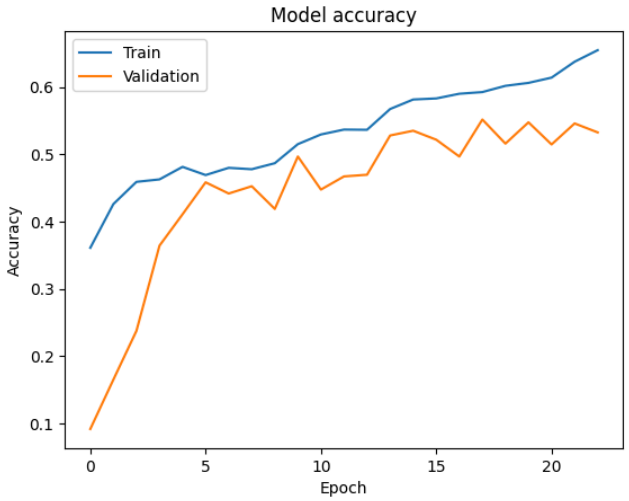
Genre	Precision	Recall	F-1	Support
Hip Hop	0.50	0.77	0.60	215
Pop	0.24	0.17	0.20	207
Rock	0.28	0.14	0.18	203
Jazz	0.74	0.71	0.72	228
Classical	0.68	0.87	0.76	183
Electronic	0.71	0.59	0.65	228
Reggae	0.37	0.36	0.36	211
Country	0.26	0.27	0.26	200
Blues	0.35	0.29	0.32	191
Folk	0.29	0.32	0.31	199
New Age	0.59	0.47	0.53	133
Metal	0.58	0.80	0.67	201
Acc			0.46	2399



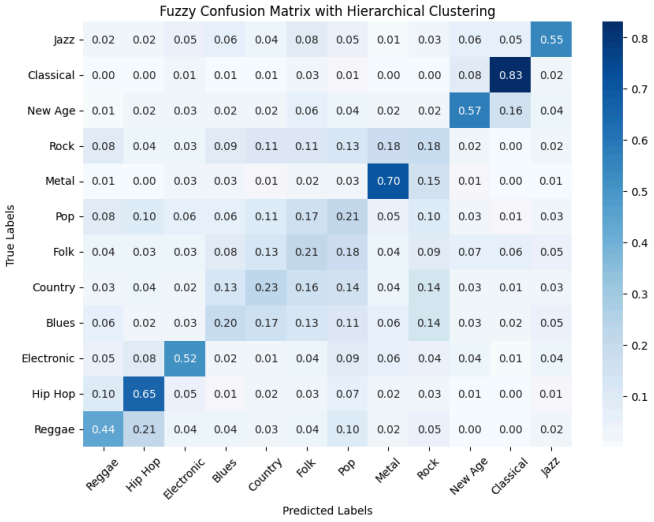
Model 2

Same feature set as model 1. Stopped early after 22 epochs in 15 minutes. 3.1 million params





Genre	Precision	Recall	F-1	Support
Hip Hop	0.58	0.83	0.69	215
Pop	0.27	0.27	0.27	207
Rock	0.33	0.23	0.27	203
Jazz	0.84	0.61	0.70	228
Classical	0.76	0.89	0.82	183
Electronic	0.77	0.68	0.72	228
Reggae	0.70	0.57	0.63	211
Country	0.37	0.43	0.40	200
Blues	0.40	0.28	0.33	191
Folk	0.35	0.33	0.34	199
New Age	0.54	0.64	0.58	133
Metal	0.63	0.88	0.73	201
Acc			0.55	2399



Analysis of Results

The first model demonstrates an ability to classify genres with varying degrees of success. Notably, genres such as Jazz, Classical, Electronic, Hip Hop, and Metal exhibit higher precision and recall rates, indicating the model's proficiency in recognizing their distinct characteristics. These genres, often characterized by unique elements such as instrumentation or rhythmic patterns, provide the model with discernible patterns to learn from, as evidenced by their higher F1-scores.

Conversely, genres like New Age, Reggae, Country, Blues, Pop, and Folk present a more complex challenge. The model's consistent confusion between New Age and Classical music could stem from their shared use of harmonic structures and similar pacing, which may not be as pronounced in the feature space. Similarly, the overlap between Reggae and Hip Hop might be attributed to shared rhythmic elements and cultural influences that are reflected in their musical composition. Interestingly, the confusion is not symmetric; for instance, Classical is not often misclassified as New Age, suggesting a unidirectional confusion. This asymmetry could be due to a disproportionate representation of defining characteristics in the dataset or an imbalance in the model's sensitivity to the features that distinguish these genres.

A closer inspection of misclassifications within the Country, Blues, Pop, and Folk genres reveals a 'block' of confusion, where these genres are frequently interchanged. This block may highlight a shared musical heritage, with common elements such as chord progressions and narrative lyrical content. However, the error analysis suggests that the model might benefit from an improved feature extraction process that can better capture and differentiate the subtle nuances between these genres.

The hierarchical clustering applied to the 'fuzzy' confusion matrix reveals insights into the model's perception of genre similarities, which could be leveraged to refine the model's architecture and training process. The clustering outcomes could guide a more targeted approach to feature selection, potentially enhancing the model's discriminative power.

The refined architecture of the updated model has demonstrated significant improvements in its ability to classify music genres, as evidenced by the increased metrics across the board. The overall accuracy has seen a substantial jump from 46% to 55%. Notably, genres such as Hip Hop and Metal have shown significant enhancements in both precision and recall, which has contributed to higher F-1 scores, suggesting a balanced improvement in the model’s confidence and its coverage of the positive class instances. The introduction of multi-scale temporal processing appears to be pivotal in capturing intricate patterns within the audio, leading to these advancements. Furthermore, the rise in precision for genres like Jazz and Reggae indicates a more refined prediction capability, whereas the increased recall for Classical and Metal suggests the model’s improved inclusiveness for these categories. These outcomes highlight the success of the architectural and processing enhancements in capturing the complex nuances necessary for effective music genre classification.

Conclusion

Reflecting on the model's real-world applicability, its strengths in identifying distinct genres like Metal and Electronic suggest utility in streaming services where clear-cut recommendations are beneficial. However, its current challenges with genre ambiguity indicate the need for further development before deployment in scenarios requiring fine-grained genre distinctions, such as in digital archiving by musicologists.

In comparison to human classification, the model aligns with the inherent subjectivity and ambiguity of genre distinctions. While a human expert might rely on a broad spectrum of cultural and historical knowledge to classify music, the model depends on the quantitative patterns within the data, which may not fully encapsulate the breadth of human experience with music.

Future work will focus on addressing the identified limitations through dataset augmentation, advanced feature extraction methods, and exploring ensemble models that combine the strengths of different machine learning approaches. The ultimate goal is to develop a robust system that not only matches human performance in clarity of genre classification but also reflects the rich and complex nature of music itself.

Appendix

Model 1 Code

```
frame_count = 431
feature_count = 24 * features_2d_count + features_1d_count
single_feature_output_size = 16
image_feature_output_size = 32

def ConvLayer(filters, kernel_size, input_layer, pool_size = None, pad_type=None):
    padding = 'valid'
    if pad_type == 'only_time':
        pad_needed = kernel_size[0] - 1
        pad_l = pad_needed // 2
        pad_r = pad_needed - pad_l
        input_layer = ZeroPadding2D(padding=((pad_l, pad_r), (0, 0)))(input_layer)
        padding = 'valid'
    elif pad_type == 'same':
        padding = 'same'

    conv = Conv2D(
        filters, kernel_size=kernel_size, activation='relu', padding=padding,
        kernel_regularizer=l2(0.001)
    )(input_layer)
```

```
    if pool_size:
        pool = MaxPooling2D(pool_size=pool_size)(conv)
        return pool
    return conv
def CompressAndReshape(input_layer, output_size):
    compressed = Conv2D(
        output_size, kernel_size=(1, 1), activation='relu',
        kernel_regularizer=l2(0.001)
    )(input_layer)

    reshaped = Reshape((input_layer.shape[1], -1))(compressed)
    return reshaped







def image_branch(input_features):   #input size: shape: (batch, 431, 24, 1)

    conv1 = ConvLayer(32, (3, 3), input_features, (2, 2), 'same')       #shape: (batch, 215, 12, 32)
    conv2 = ConvLayer(64, (3, 3), conv1, (2, 2), 'same')                #shape: (batch, 107, 6, 64)
    conv3 = ConvLayer(128, (6, 6), conv2, (2, 1), 'only_time')          #shape: (batch, 53, 1, 128)

    reshaped = CompressAndReshape(conv3, image_feature_output_size)   #shape: (batch, 53, 48)

    return reshaped

def single_branch(input_features): #input size: (batch, 431, features_1d_count, 1)

    conv1 = ConvLayer(32, (3, features_1d_count), input_features, (2, 1), 'only_time')
        #shape: (batch, 215, 1, 32)
    conv2 = ConvLayer(64, (3, 1), conv1, (2, 1), 'only_time')
        #shape: (batch, 107, 1, 64)
    conv3 = ConvLayer(128, (6, 1), conv2, (2, 1), 'only_time')
        #shape: (batch, 53, 1, 128)

    reshaped = CompressAndReshape(conv3, single_feature_output_size)
        #shape: (batch, 53, 32)

    return reshaped

input_combined = Input(shape=(frame_count, feature_count))
input_combined = Reshape((frame_count, feature_count, 1))(input_combined)

features_2d_lambdas = []
single_feature_begin = 0
for i in range(features_2d_count):
    features_2d_lambdas.append(Lambda(lambda x, pos=i: x[:, :, 24 * pos:24 * (pos + 1),
:])(input_combined))
    if i == features_2d_count - 1:
        single_feature_begin = 24 * (i + 1)

single_features = Lambda(lambda x: x[:, :, single_feature_begin:, :])(input_combined)
    #shape: (batch, 431, 6, 1)

processed_2d_features = []
for feature in features_2d_lambdas:
```

```
    processed_2d_features.append(image_branch(feature))

single_reshaped = single_branch(single_features)

#concatenate the branches
combined = Concatenate()(processed_2d_features + [single_reshaped])
    #shape: (batch, 53, ~)

#LSTM layer
lstm_1 = LSTM(
    256, return_sequences=True,
    kernel_regularizer=l2(0.001)
)(combined)                                 #shape: (batch, 51, 256)
lstm_1 = Dropout(0.5)(lstm_1)

lstm_output = LSTM(
    128, return_sequences=False,
    kernel_regularizer=l2(0.001)
)(lstm_1)                                   #shape: (batch, 128)
lstm_output = Dropout(0.3)(lstm_output)

#Dense layer
dense_1 = Dense(128, activation='relu')(lstm_output) #shape: (batch, 128)
dense_1 = Dropout(0.3)(dense_1)

output_layer = Dense(num_genres, activation='softmax')(dense_1) #shape: (batch, num_genres)

model = Model(inputs=input_combined, outputs=output_layer)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

Model 2 Code

```
frame_count = 431
feature_count = 24 * features_2d_count + features_1d_count

def ConvLayer(filters, kernel_size, input_layer, pool_size = None, pad_type=None):

    padding = 'valid'
    if pad_type == 'only_time':
        pad_needed = kernel_size[0] - 1
        pad_l = pad_needed // 2
        pad_r = pad_needed - pad_l
        input_layer = ZeroPadding2D(padding=((pad_l, pad_r), (0, 0)))(input_layer)
        padding = 'valid'
    elif pad_type == 'same':
        padding = 'same'

    conv = Conv2D(
        filters, kernel_size=kernel_size, activation='relu', padding=padding,
        kernel_regularizer=l2(0.001)
    )(input_layer)

    conv = BatchNormalization()(conv)

    if pool_size:
        pool = MaxPooling2D(pool_size=pool_size)(conv)
        return pool

    return conv

def image_branch(input_features):   #input_features shape: (batch, 431, 24, 1)

    conv1 = ConvLayer(64, (3, 3), input_features, (2, 2), 'same')       #shape: (batch, 215, 12, 64)
    conv2 = ConvLayer(64, (3, 3), conv1, (2, 2), 'same')                #shape: (batch, 107, 6, 64)
    conv3 = ConvLayer(128, (3, 3), conv2, (2, 2), 'same')               #shape: (batch, 53, 3, 128)
    conv4 = ConvLayer(256, (3, 3), conv3, pool_size=None, pad_type='only_time')          #shape: (batch,
53, 1, 256)

    conv_reduced = Conv2D(64, (1, 1), activation='relu', kernel_regularizer=l2(0.001))(conv4)   #shape:
(batch, 53, 1, 64)
    conv_reduced = BatchNormalization()(conv_reduced)

    reshaped = Reshape((53, 64))(conv_reduced)

    return reshaped
```

```
def single_branch(input_features):

    conv1 = ConvLayer(64, (3, features_1d_count), input_features, (2, 1), 'only_time')
        #shape: (batch, 215, 6, 32)
    conv2 = ConvLayer(64, (3, 1), conv1, (2, 1), 'only_time')
        #shape: (batch, 107, 6, 64)
    conv3 = ConvLayer(128, (3, 1), conv2, (2, 1), 'only_time')
        #shape: (batch, 53, 6, 128)
    conv4 = ConvLayer(256, (6, 1), conv3, pad_type='only_time')
        #shape: (batch, 53, 1, 128)

    conv_reduced = Conv2D(64, (1, 1), activation='relu', kernel_regularizer=l2(0.001))(conv4)
        #shape: (batch, 53, 1, 32)
    conv_reduced = BatchNormalization()(conv_reduced)

    reshaped = Reshape((53, 64))(conv_reduced)

    return reshaped

def LSTM_layer(input_features):

    #LSTM layer
    lstm_1 = LSTM(
        128, return_sequences=True,
        kernel_regularizer=l2(0.005)
    )(combined)                                     #shape: (batch, 51, 128)
    lstm_1 = Dropout(0.3)(lstm_1)

    attn = Attention()([lstm_1, lstm_1])

    lstm_output = LSTM(
        64, return_sequences=False,
        kernel_regularizer=l2(0.005)
    )(attn)                                       #shape: (batch, 64)
    lstm_output = Dropout(0.3)(lstm_output)

    return lstm_output

def down_sample_features_conv(input_features, ratio=2):
    #expects input shape (batch, time_steps, 1, features)
    num_features = input_features.shape[-1]
    return Conv2D(num_features, (ratio, 1), strides=(ratio, 1), padding='valid',
        activation='relu')(input_features)
```

```
input_combined = Input(shape=(frame_count, feature_count))
input_combined = Reshape((frame_count, feature_count, 1))(input_combined)

vector_feature_lambdas = []
single_feature_begin = 24 * features_2d_count

for i in range(features_2d_count):
    vector_feature_lambdas.append(
        Lambda(lambda x, pos=i: x[:, :, 24 * pos:24 * (pos + 1), :])(input_combined)
    )

scalar_features = Lambda(lambda x: x[:, :, single_feature_begin:, :])(input_combined)

processed_features = []
for vector_feature in vector_feature_lambdas:
    #each 2d branch outputs a tensor with shape (batch, 53, 32)
    processed_features.append(image_branch(vector_feature))

#single branch outputs a tensor with shape (batch, 53, 32)
processed_features.append(single_branch(scalar_features))

#Primary Decision Branch
#concatenate the branches, features: 5 x 64 = 320
combined = Concatenate()(processed_features)     #shape: (batch, 53, 320)

combined_1 = Reshape((53, 1, 320))(combined)    #shape: (batch, 53, 1, 320)
combined_2 = down_sample_features_conv(combined_1, 2)    #shape: (batch, 26, 1, 320)
combined_3 = down_sample_features_conv(combined_2, 2)    #shape: (batch, 13, 1, 320)
combined_4 = down_sample_features_conv(combined_3, 2)    #shape: (batch, 6, 1, 320)

lstm_1 = LSTM_layer(combined_1)
lstm_2 = LSTM_layer(combined_2)
lstm_3 = LSTM_layer(combined_3)
lstm_4 = LSTM_layer(combined_4)

combined_lstm = Concatenate()([lstm_1, lstm_2, lstm_3, lstm_4]) #shape: (batch, 256)

#Dense Layer
dense = Dense(128, activation='relu', kernel_regularizer=l2(0.001))(combined_lstm)
dense = Dropout(0.6)(dense)

softmax = Dense(num_genres, activation='softmax')(dense)

model = Model(inputs=input_combined, outputs=softmax)
model.compile(optimizer=tf.keras.optimizers.Adam(), loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.summary()
```