Music Genre Classification

Kirtan Pokiya

1. Introduction

Music genre classification is a fundamental task in audio content analysis with applications spanning music recommendation, playlist generation, and music retrieval systems. Despite significant advancements in machine learning and signal processing techniques, accurately categorizing music into distinct genres remains a challenging endeavor due to the subjective nature of genre boundaries and the complexity of musical features. This paper aims to address this challenge by proposing a novel approach to music genre classification that leverages deep learning architectures and feature extraction methods to achieve improved classification accuracy and robustness.

1.1. Purpose of the Project

The purpose of this project is to develop and evaluate a robust music genre classification system using Deep learning techniques. By accurately categorizing music into various genres, the system aims to enhance user experience in music streaming platforms and facilitate efficient music organization and retrieval

1.2. Significance of the Project

The significance of this project lies in its potential to revolutionize the way we interact with music, offering personalized recommendations and streamlined music organization based on genre classification. By automating the genre classification process, this system can save time and effort for users and music professionals alike, ultimately enriching the music listening experience.

1.3. Research Questions

- How effective are machine learning algorithms in classifying music genres, and what factors contribute to their performance, such as feature selection, model architecture, and dataset characteristics?

1.4. Description of the Dataset

The dataset used in this project, sourced from Kaggle:

GTZAN Genre Collection

(https://www.kaggle.com/datasets/carlthome/gtzan-genre-collection)

The dataset consists of 1000 audio tracks each 30 seconds long. It contains 10 genres (folders), each represented by 100 tracks.

The tracks are all 22050 Hz monophonic 16-bit audio files in .au format.

genres = {'blues': 0, 'classical': 1, 'country': 2, 'disco': 3, 'hiphop': 4, 'jazz': 5, 'metal': 6, 'pop': 7, 'reggae': 8, 'rock': 9}

2. Data Preprocessing

2.1. Data Preparation: The main step was to convert the Audio data (.au file) into something which can be read later on by our machine learning or deep learning models. For this Librosa Library was used.

This Librosa library converts the Audio data into the Mel spectrogram of that song. Mel spectrogram represents the loudness of each frequency of the sound in a time series.

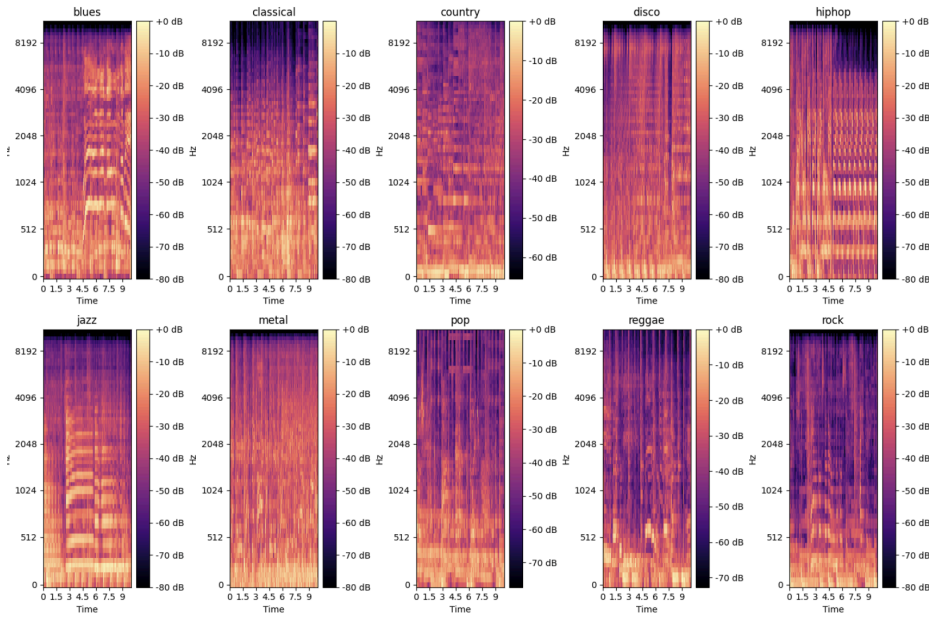
```
y, sr = librosa.load(songname, mono=True, duration=30)  # Load the entire 30-second au

# Split the audio into segments
num_segments = len(y) // (segment_duration * sr)
for index in range(num_segments):
    start_sample = index * segment_duration * sr
    end_sample = start_sample + segment_duration * sr
    segment = y[start_sample:end_sample]

    # Extract Mel spectrogram features
    ps = librosa.feature.melspectrogram(y=segment, sr=sr, hop_length=hop_length, n_fft
    ps_db = librosa.power_to_db(ps, ref=np.max)

    # Append features and label to the dataset
    dataset.append((ps_db, label))
```

2.2. Exploratory Data Analysis and Visualization

```
dataset[0]
```

```
(array([[-22.673328 , -14.474144 , -21.120438 , ..., -20.38428  ,
         -19.402018 , -22.727526 ],
        [-23.606817 , -12.7264595, -18.015732 , ..., -18.1247   ,
         -14.115858 , -22.387379 ],
        [-26.272598 , -17.164982 , -17.758533 , ..., -20.331182 ,
         -15.655169 , -25.771301 ],
        ...,
        [-59.159443 , -59.509483 , -63.52632  , ..., -56.84107  ,
         -62.736008 , -61.431686 ],
        [-67.1499   , -73.02876  , -75.86813  , ..., -73.11742  ,
         -72.35505  , -70.37918  ],
        [-67.65089  , -80.       , -80.       , ..., -80.       ,
         -80.       , -71.925446 ]], dtype=float32),
 0)
```

The numpy array represents the Mel spectrogram of the song and the last element, 0 in this case, represents the label or genre of the song.



2.3. Data Splitting

To facilitate model training and evaluation, we split the dataset into training and testing subsets, ensuring an optimal balance between model performance and generalization with 20 percent as our testing data.

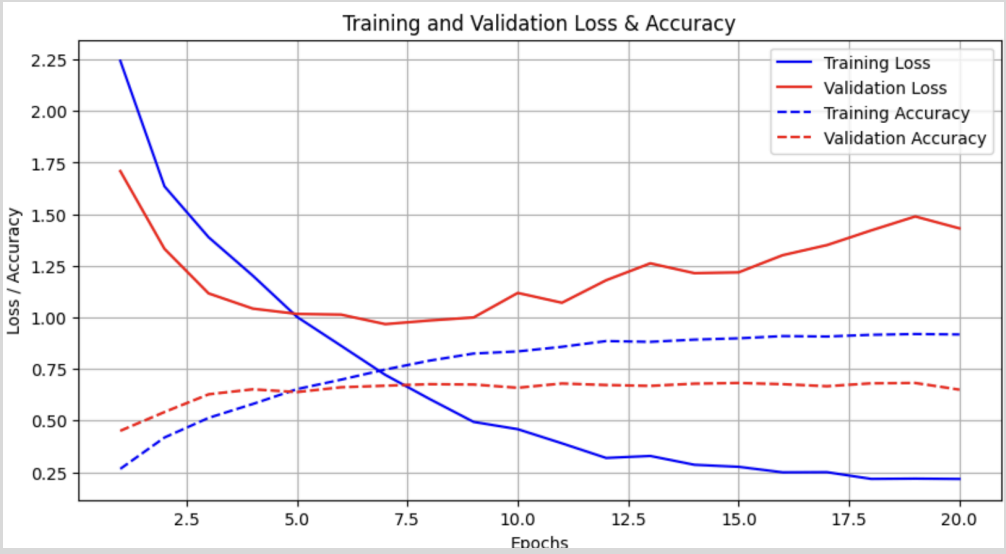
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_onehot, test_size=0.2, random_state=42)
```

3. Model Building and Evaluation

3.1. Model Building

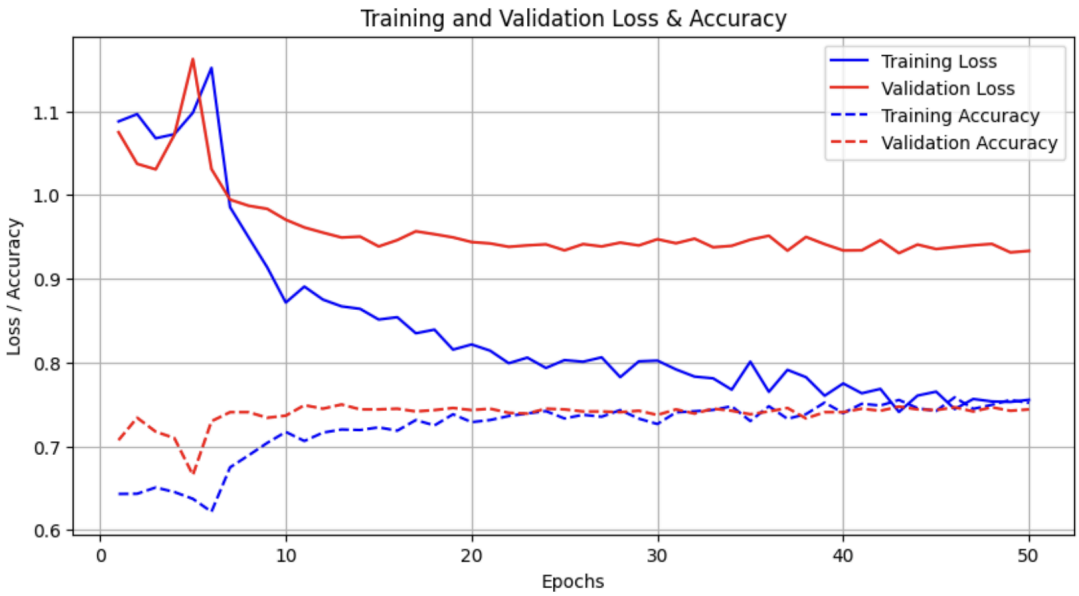
We embarked on model building with three distinct approaches:

Simple Convolutional Neural Network (CNN) : Serving as the baseline, this model does not yield promising results with a test accuracy of 65 percent. Also the model was overfitting, with the training accuracy being nearly perfect.




```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(X_train.shape[1:])))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, batch_size=32, epochs=20,
validation_data=(X_test, y_test)
```

Transfer Learning (VGG16): Leveraging transfer learning techniques, this model exhibited improved performance with test accuracy of .




```
base_model = VGG16(weights='imagenet', include_top=False,
input_shape=X_train.shape[1:])

# Freeze the convolutional layers
for layer in base_model.layers:
    layer.trainable = False

# Add fully connected layers on top of the base model
model = Sequential()
model.add(base_model)
model.add(Flatten())
model.add(Dense(128, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)))  # Adding L2 regularization
model.add(Dropout(0.5))  # Adding dropout
model.add(Dense(10, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3,
min_lr=0.0001)

# Train the model
```

```
history = model.fit(X_train, y_train, batch_size=32, epochs=50,
validation_data=(X_test, y_test), callbacks=[reduce_lr])
```

Non Deep Learning Model (Random Forest Model): This model completely failed to classify the music genres with test accuracy of just over 50 percent.

```
# Flatten each spectrogram
X_flat = np.array([spectrogram.flatten() for spectrogram in X])

# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_flat, y, test_size=0.2,
random_state=42)

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the Random Forest classifier
rf_classifier.fit(X_train, y_train)

# Predict labels for test data
y_pred = rf_classifier.predict(X_test)
```

4. Conclusion

In conclusion, The outcome of the VGG16 model was good and can be used to classify the music genres. The Non-deep learning models are not deep enough to understand the complexity of the dataset and hence cannot be used.

Also in order to make our models more efficient this techniques can be used:

Hyperparameter Tuning: Fine-tune parameters such as learning rate, batch size, and optimizer choice to optimize model performance. This involves experimenting with different values to find the combination that yields the best results

Ensemble Methods: Combine predictions from multiple models to improve accuracy and reduce errors. This can involve training several models with different initializations or architectures and averaging their predictions.

Exploring Different Architectures: Experiment with various neural network architectures, such as deeper or wider networks, or even different types of models altogether, to find the one that best suits the problem domain.