Music genre classification using Deep Learning
Final Project

INTRODUCTION

Music is a big part of our lives and with the vast number of genres available, automatically classifying music into genres can be useful for many applications, such as music recommendation systems.

In this project, I used the **GTZAN** dataset from *Kaggle, which contains audio samples from 10 different genres,
Blues - Classical - Country - Disco - Hiphop - Jazz - Metal - Pop - Reggae - Rock.
Each audio file is 30 seconds long, providing a rich source of data for genre classification.
I built two deep learning models to classify these genres based on audio features.

Along with the audio files, the dataset also includes **Mel Spectrograms**, which are visual representations of the sound. Alternatively, we could use CNNs to classify the genres, as the spectrograms transform the audio into an image-like format.

I will present the steps I took, from data exploration and feature extraction to building and evaluating the models.

*https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification

Libraries

```
import os
import numpy as np
import librosa
import librosa.display
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from tensorflow.keras import layers, models, optimizers, Input
from tensorflow.keras.callbacks import EarlyStopping
import optuna
import warnings
```

Librosa - Optuna

Librosa is a great python package for music and audio analysis.

Documentation on librosa can be found here:

https://librosa.org/doc/latest/index.html

Optuna also offers good performances in hyperparameter tuning.

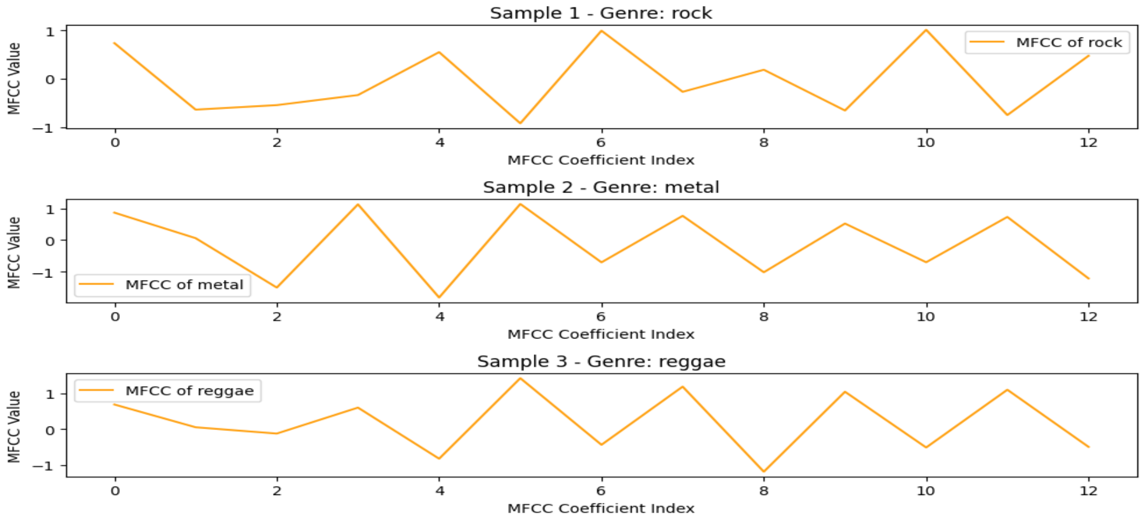
Documentation on optuna can be found here:

https://optuna.org/

Exploratory Data Analysis Visualizations Part 1

```python
def plot_training_audio_samples(X_train, y_train, num_samples = 5):
    plt.figure(figsize=(10, 10))
    for i in range(num_samples):
        idx = np.random.randint(0, len(X_train))
        mfcc = X_train[idx]
        genre = label_encoder.inverse_transform([y_train[idx]])[0]

        plt.subplot(5, 1, i + 1)
        plt.plot(mfcc, label=f"MFCC of {genre}", color = "orange")
        plt.title(f"Sample {i + 1} - Genre: {genre}")
        plt.xlabel("MFCC Coefficient Index")
        plt.ylabel("MFCC Value")
        plt.legend()
    plt.tight_layout()
    plt.show()

plot_training_audio_samples(X_train, y_train)
```

MFCC visualization



MFCC (Mel-Frequency Cepstral Coefficients) is a feature commonly used in audio processing.

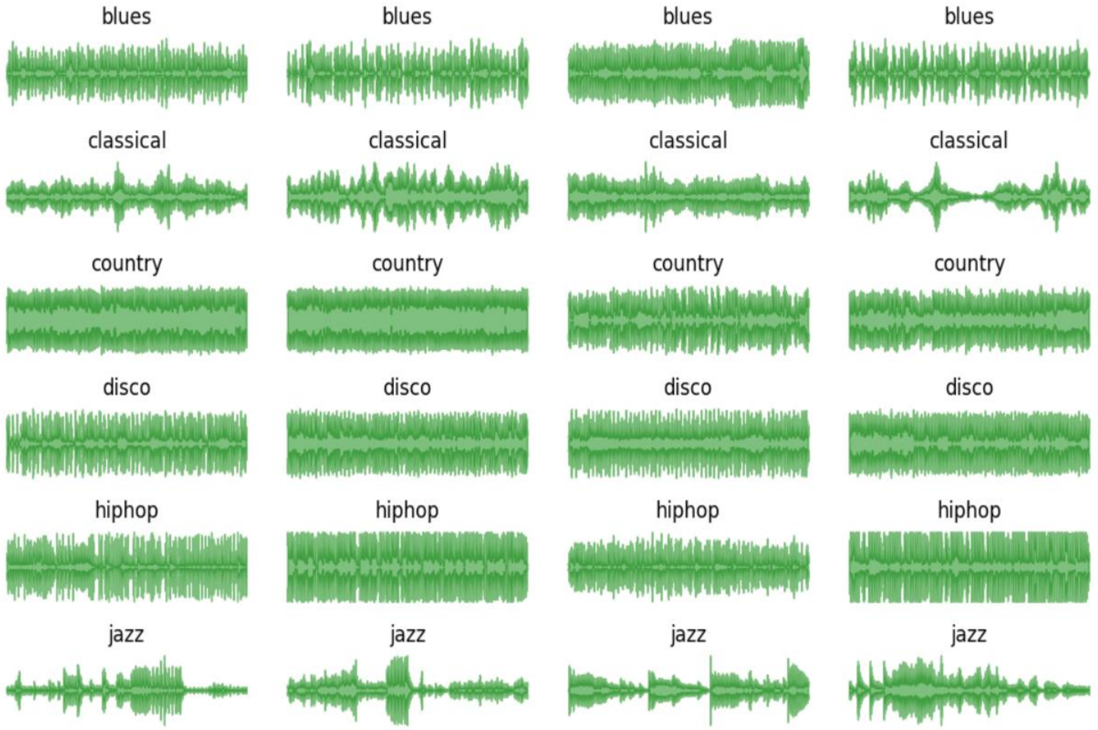
The MFCC captures the short-term power spectrum of a sound signal.

See https://en.wikipedia.org/wiki/Mel-frequency_cepstrum for more information.

Exploratory Data Analysis Visualizations Part 2

```
def plot_audio_samples(genres, num_samples=4):
    plt.figure(figsize=(10, 10))
    for genre in genres:
        genre_dir = os.path.join(data_directory, genre)
        files = os.listdir(genre_dir)
        for i in range(num_samples):
            file_path = os.path.join(genre_dir, files[i])
            y, sr = librosa.load(file_path, sr=22050)
            plt.subplot(len(genres), num_samples, genres.index(genre) * num_samples + i + 1
            plt.title(genre)
            librosa.display.waveshow(y, sr=sr, alpha=0.5, color = "green")
            plt.axis("off")
    plt.tight_layout()
    plt.show()
```

Waveshow with Librosa



Exploratory Data Analysis
Music Audio Samples

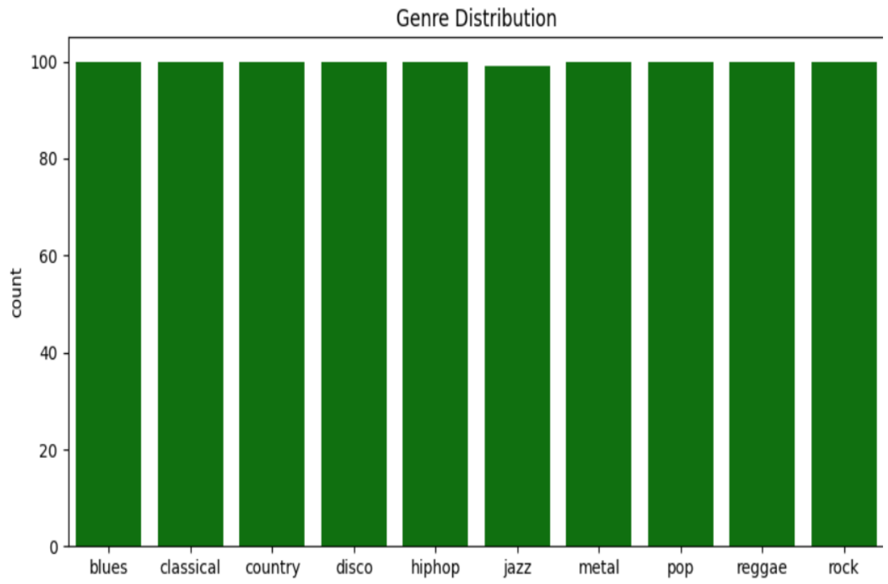
```
from IPython.display import Audio

country_sample_path = r"/kaggle/input/gtzan-dataset-music-genre-classification/Data/genres_original\country\country.00003.wav"
pop_sample_path = r"/kaggle/input/gtzan-dataset-music-genre-classification/Data/genres_original\pop\pop.00010.wav"

print("Country Music Sample:")
country_audio = Audio(country_sample_path, autoplay=True)
display(country_audio)

print("Pop Music Sample:")
pop_audio = Audio(pop_sample_path, autoplay=True)
display(pop_audio)
```

Country and Pop audio samples

Exploratory Data Analysis Classes

```
warnings.simplefilter(action = "ignore", category = FutureWarning)

data_directory = r"/kaggle/input/gtzan-dataset-music-genre-classification/Data/genres_original"
genres = "blues classical country disco hiphop jazz metal pop reggae rock".split()

audio_data = []
labels = []

for genre in genres:
    genre_dir = os.path.join(data_directory, genre)
    for file in os.listdir(genre_dir):
        if file.endswith(".wav"):
            file_path = os.path.join(genre_dir, file)
            try:
                y, sr = librosa.load(file_path, sr=22050)
                mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
                mfccs = np.mean(mfccs.T, axis=0)
                audio_data.append(mfccs)
                labels.append(genre)
            except Exception as e:
                print(f"file may be corrupted {file_path}: {e}")
                continue

plt.figure(figsize=(10, 5))
sns.countplot(x=labels, color="green")
plt.title("Genre Distribution")
plt.show()
```

Classes are perfectly balanced



Exploratory Data Analysis
Data Preparation

```
audio_data = np.array(audio_data)
labels = np.array(labels)

label_encoder = LabelEncoder()
y = label_encoder.fit_transform(labels)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(audio_data)

X_train, X_val, y_train, y_val = train_test_split(X_scaled, y, test_size = 0.2, random_state = 42, stratify = y)

hyperparameter_range = {
    "learning_rate": (1e-7, 1e-3),
    "batch_size": [8, 16, 32, 64],
    "dropout_rate": (0.1, 0.4)
}

total_epochs = 100
n_trials = 10
batch_size_alternative = 16
```

Data preparation

- Data cleaned

  Just one jazz file seemed corrupted

- Data encoded and scaled

- Hyperparameters range defined

Model Architecture Definition

```
def build_model(learning_rate, dropout_rate):
    model = models.Sequential()
    model.add(Input(shape=(X_scaled.shape[1],)))
    model.add(layers.Dense(256, activation="relu"))
    model.add(layers.Dropout(dropout_rate))
    model.add(layers.Dense(128, activation="relu"))
    model.add(layers.Dropout(dropout_rate))
    model.add(layers.Dense(64, activation="relu"))
    model.add(layers.Dense(len(genres), activation="softmax"))

    optimizer = optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss="sparse_categorical_crossentropy", metrics=["accuracy"])
    return model

def build_alternative_model(learning_rate):
    model = models.Sequential()
    model.add(Input(shape=(X_scaled.shape[1],)))
    model.add(layers.Dense(128, activation="relu"))
    model.add(layers.Dense(len(genres), activation="softmax"))

    optimizer = optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss="sparse_categorical_crossentropy", metrics=["accuracy"])
    return model
```

One complex and one simple model

Model Architecture
Training and Optimization Part 1

```
def objective(trial):

    learning_rate = trial.suggest_loguniform("learning_rate", *hyperparameter_range["learning_rate"])
    batch_size = trial.suggest_categorical("batch_size", hyperparameter_range["batch_size"])
    dropout_rate = trial.suggest_uniform("dropout_rate", *hyperparameter_range["dropout_rate"])

    model = build_model(learning_rate, dropout_rate)

    early_stop = EarlyStopping(monitor="val_loss", patience =33 , restore_best_weights=True)

    history = model.fit(X_train, y_train, validation_data=(X_val, y_val),
                        epochs=total_epochs, batch_size=batch_size,
                        callbacks=[early_stop], verbose=0)

    return max(history.history["val_accuracy"])

study = optuna.create_study(direction="maximize", study_name="music_classification")
study.optimize(objective, n_trials=n_trials)

best_trial = study.best_trial
print("Best hyperparameters for the complex model:")
print(f"Learning Rate: {best_trial.params['learning_rate']}")
print(f"Batch Size: {best_trial.params['batch_size']}")
print(f"Dropout Rate: {best_trial.params['dropout_rate']}")
print(f"Best trial validation accuracy: {best_trial.value}")
```

Optuna optimization

Model Architecture
Training and Optimization Part 2

```
best_params = best_trial.params
model_complex = build_model(learning_rate=best_params["learning_rate"], dropout_rate=best_params["dropout_rate"])

callbacks = []
early_stop = EarlyStopping(monitor="val_loss", patience=5, restore_best_weights=True)
callbacks.append(early_stop)

history_complex = model_complex.fit(X_train, y_train, validation_data=(X_val, y_val),
                    epochs=total_epochs, batch_size=best_params["batch_size"],
                    callbacks=callbacks)

learning_rate_alternative = best_params["learning_rate"]
model_alternative = build_alternative_model(learning_rate_alternative)

history_alternative = model_alternative.fit(
    X_train,
    y_train,
    validation_data=(X_val, y_val),
    epochs=total_epochs,
    batch_size = batch_size_alternative,
    callbacks=[early_stop],
    verbose=0
)

model_complex.save("music_classification.keras")
```

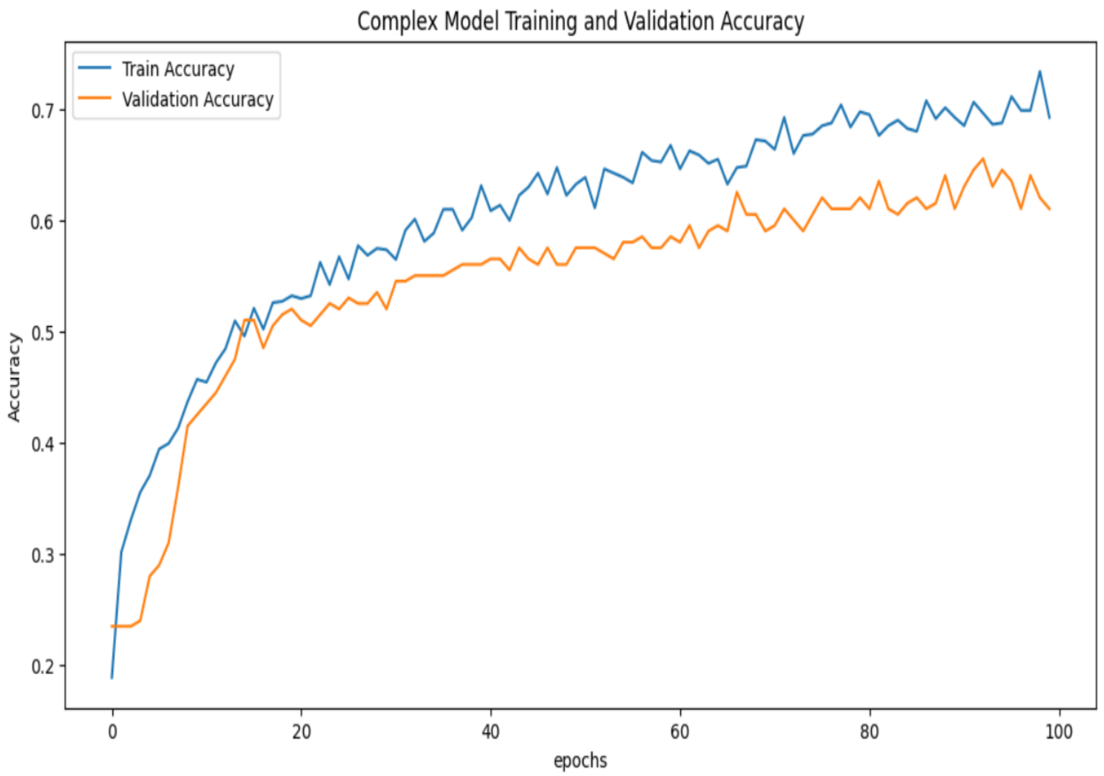
Best hyperparameters

Models trained on best hyperparameters from optimization

Model Architecture
Training History

```
def plot_training_history(history, model_type):
    plt.figure(figsize=(12, 6))
    plt.plot(history.history["accuracy"], label="Train Accuracy")
    plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
    plt.title(f"{model_type} Training and Validation Accuracy")
    plt.xlabel("epochs")
    plt.ylabel("Accuracy")
    plt.legend()
    plt.show()

    plt.figure(figsize=(12, 6))
    plt.plot(history.history["loss"], label="Train Loss")
    plt.plot(history.history["val_loss"], label="Validation Loss")
    plt.title(f"{model_type} Training and Validation Loss")
    plt.xlabel("epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()

plot_training_history(history_complex, "Complex Model")
plot_training_history(history_alternative, "Alternative Model")
```

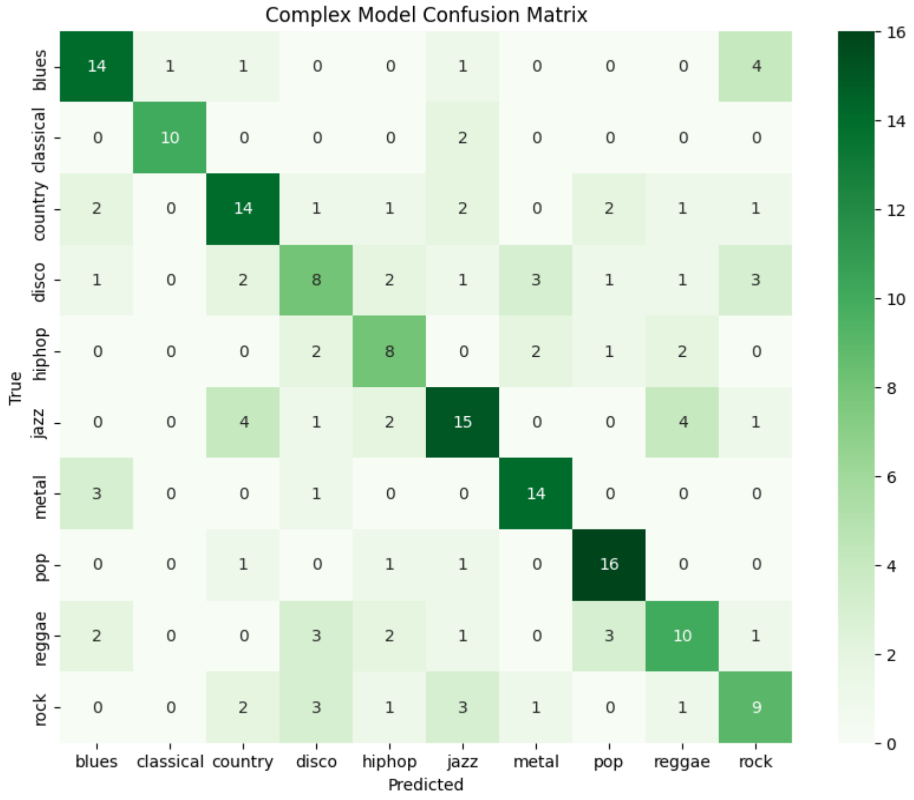
Best model training history



Model Performance Confusion Matrix

```
def plot_confusion_matrix(y_val, y_pred_classes, model_type, cmap):
    cm = confusion_matrix(y_val, y_pred_classes)
    plt.figure(figsize=(10, 8))
    sns.heatmap(cm, annot=True, fmt="d", cmap = cmap, xticklabels=genres, yticklabels=genres)
    plt.title(f"{model_type} Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.show()

y_pred_complex = model_complex.predict(X_val)
y_pred_classes_complex = np.argmax(y_pred_complex, axis=1)
plot_confusion_matrix(y_val, y_pred_classes_complex, "Complex Model", cmap = "Greens")

y_pred_alternative = model_alternative.predict(X_val)
y_pred_classes_alternative = np.argmax(y_pred_alternative, axis=1)
plot_confusion_matrix(y_val, y_pred_classes_alternative, "Alternative Model", cmap = "Blues")
```

Best model confusion matrix



Performance Summary

```
def evaluate_model(model, X_val, y_val):
    y_pred = model.predict(X_val)
    y_pred_classes = np.argmax(y_pred, axis=1)
    accuracy = accuracy_score(y_val, y_pred_classes)
    return accuracy, classification_report(y_val, y_pred_classes, target_names=genres)

accuracy_complex, report_complex = evaluate_model(model_complex, X_val, y_val)
accuracy_alternative, report_alternative = evaluate_model(model_alternative, X_val, y_val)

print("Complex Model Evaluation:")
print(f"Validation Accuracy: {accuracy_complex:.4f}")
print(report_complex)

print("\nAlternative Model Evaluation:")
print(f"Validation Accuracy: {accuracy_alternative:.4f}")
print(report_alternative)
```

Evaluation Report

```
Complex Model Evaluation:
Validation Accuracy: 0.5900
              precision    recall  f1-score   support

       blues       0.64      0.67      0.65        21
   classical       0.91      0.83      0.87        12
     country       0.58      0.58      0.58        24
       disco       0.42      0.36      0.39        22
      hiphop       0.47      0.53      0.50        15
        jazz       0.58      0.56      0.57        27
       metal       0.70      0.78      0.74        18
         pop       0.70      0.84      0.76        19
      reggae       0.53      0.45      0.49        22
        rock       0.47      0.45      0.46        20

    accuracy                           0.59       200
   macro avg       0.60      0.61      0.60       200
weighted avg       0.59      0.59      0.59       200
```

CONCLUSION

The overall validation accuracy achieved is 59%, with varying performance across different genres.

Classical and pop genres show the highest F1-scores of 0.87 and 0.76, respectively, indicating better classification precision and recall for these categories.

However, genres such as disco and reggae exhibit lower F1-scores, with values of 0.39 and 0.49, suggesting that the model struggles with these genres.

The precision-recall balance is reasonable across some categories but inconsistent in others, such as hiphop and disco, where both scores are relatively low.

In conclusion, while the model demonstrates potential in classifying certain music genres, improvements in handling underrepresented or more challenging genres are needed to enhance overall performance.

The dataset also includes Mel Spectrograms, which are visual representations of the songs.
A potential next step could be to build a model based on these spectrograms and compare its performance with the model using audio features.