In [1]:

```
import os
import librosa
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
```

```
C:\Users\shrut\anaconda3\lib\site-packages\numpy\_distributor_init.py:30: Use
rWarning: loaded more than 1 DLL from .libs:
C:\Users\shrut\anaconda3\lib\site-packages\numpy\.libs\libopenblas.GK7GX5KEQ4
F6UYO3P26ULGBQYHGQO7J4.gfortran-win_amd64.dll
C:\Users\shrut\anaconda3\lib\site-packages\numpy\.libs\libopenblas.WCDJNK7YVM
PZQ2ME2ZZHJJRJ3JIKNDB7.gfortran-win_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
C:\Users\shrut\anaconda3\lib\site-packages\tensorflow\python\keras\engine\tra
ining_arrays_v1.py:37: UserWarning: A NumPy version >=1.22.4 and <2.3.0 is re
quired for this version of SciPy (detected version 1.21.0)
  from scipy.sparse import issparse  # pylint: disable=g-import-not-at-top
```

In [2]:

```
# Set paths
DATA_PATH = 'Audio_Files/'  # Update this with your path
major_folder = os.path.join(DATA_PATH, 'major')
minor_folder = os.path.join(DATA_PATH, 'minor')

# Hyperparameters
n_mfcc = 20
sequence_length = 30
batch_size = 32
epochs = 50
```

In [3]:

```
# Function to load and preprocess audio files
def load_and_extract_features(folder_path):
    features = []
    for file_name in os.listdir(folder_path):
        if file_name.endswith('.wav'):
            file_path = os.path.join(folder_path, file_name)
            y, sr = librosa.load(file_path)
            mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)
            features.append(mfccs.T)
    return features
```

In [4]:

```
# Load and prepare dataset
major_features = load_and_extract_features(major_folder)
minor_features = load_and_extract_features(minor_folder)

# Prepare sequences
def create_sequences(features, seq_length):
    X, y = [], []
    for mfcc in features:
        for i in range(len(mfcc) - seq_length):
            X.append(mfcc[i:i + seq_length])
            y.append(mfcc[i + seq_length])
    return np.array(X), np.array(y)
```

In [5]:

```
X_major, y_major = create_sequences(major_features, sequence_length)
X_minor, y_minor = create_sequences(minor_features, sequence_length)

# Combine and label data (1 for major, 0 for minor)
X = np.concatenate((X_major, X_minor), axis=0)
y = np.concatenate((y_major, y_minor), axis=0)
```

In [6]:

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, r
```

In [7]:

```
# Define the model
model = Sequential([
    LSTM(64, input_shape=(sequence_length, n_mfcc), return_sequences=True)
    Dropout(0.2),
    LSTM(64),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(n_mfcc)  # Output layer with the same dimension as MFCC input
])
```

```
C:\Users\shrut\anaconda3\lib\site-packages\keras\src\layers\rnn\rnn.py:204: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Whe
n using Sequential models, prefer using an `Input(shape)` object as the first
layer in the model instead.
  super().__init__(**kwargs)
```

In [*]:

```
model.compile(optimizer='adam', loss='mse')
model.summary()

# Set up early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_bes

# Train the model with early stopping
history = model.fit(
    X_train, y_train,
    batch_size=batch_size,
    epochs=epochs,
    validation_split=0.2,
    callbacks=[early_stopping]
)
```

Model: "sequential"

Layer (type)	**Output Shape**	
lstm (LSTM)	(None, 30, 64)	
dropout (Dropout)	(None, 30, 64)	
lstm_1 (LSTM)	(None, 64)	
dropout_1 (Dropout)	(None, 64)	
dense (Dense)	(None, 32)	
dense_1 (Dense)	(None, 20)	

Total params: 57,524 (224.70 KB)

Trainable params: 57,524 (224.70 KB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 52s 41ms/step - loss: 4055.2861 - val_loss: 60
6.1160
Epoch 2/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 45s 40ms/step - loss: 490.6552 - val_loss: 14
3.1201
Epoch 3/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 41s 35ms/step - loss: 192.3761 - val_loss: 10
9.3244
Epoch 4/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 40s 35ms/step - loss: 165.5600 - val_loss: 10
0.6715
Epoch 5/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 42s 36ms/step - loss: 155.7856 - val_loss: 97.
2126
Epoch 6/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 39s 34ms/step - loss: 150.0359 - val_loss: 96.
6808
Epoch 7/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 39s 34ms/step - loss: 146.2354 - val_loss: 92.
5811
Epoch 8/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 41s 35ms/step - loss: 142.7170 - val_loss: 89.
9662
Epoch 9/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 40s 35ms/step - loss: 138.0513 - val_loss: 87.
4177
Epoch 10/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 42s 37ms/step - loss: 134.7647 - val_loss: 87.
6687
Epoch 11/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 83s 37ms/step - loss: 129.9500 - val_loss: 83.
6419
Epoch 12/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 41s 36ms/step - loss: 130.1402 - val_loss: 79.
7878
Epoch 13/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 42s 36ms/step - loss: 122.5917 - val_loss: 81.
1944
Epoch 14/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 40s 35ms/step - loss: 120.8897 - val_loss: 77.
4183
Epoch 15/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 41s 36ms/step - loss: 118.4245 - val_loss: 75.
8484
Epoch 16/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 45s 39ms/step - loss: 114.2781 - val_loss: 75.
6577
Epoch 17/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 42s 37ms/step - loss: 112.3637 - val_loss: 73.
7750
Epoch 18/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 44s 38ms/step - loss: 108.2301 - val_loss: 71.
2567
Epoch 19/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 46s 40ms/step - loss: 104.5000 - val_loss: 72.
1390
```

```
Epoch 20/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 52s 45ms/step - loss: 100.4783 - val_loss: 67.
2329
Epoch 21/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 129s 86ms/step - loss: 95.1257 - val_loss: 62.
8173
Epoch 22/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 148s 91ms/step - loss: 90.4192 - val_loss: 59.
2195
Epoch 23/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 139s 88ms/step - loss: 85.2510 - val_loss: 59.
1313
Epoch 24/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 129s 76ms/step - loss: 80.3584 - val_loss: 53.
4864
Epoch 25/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 57s 49ms/step - loss: 75.3179 - val_loss: 51.2
329
Epoch 26/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 101s 88ms/step - loss: 70.8768 - val_loss: 49.
9078
Epoch 27/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 73s 63ms/step - loss: 67.9127 - val_loss: 47.8
595
Epoch 28/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 80s 61ms/step - loss: 65.4084 - val_loss: 46.7
697
Epoch 29/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 89s 78ms/step - loss: 61.8384 - val_loss: 45.6
907
Epoch 30/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 150s 84ms/step - loss: 60.6193 - val_loss: 43.
2519
Epoch 31/50
1148/1148 ━━━━━━━━━━━━━━━━━━━━ 135s 78ms/step - loss: 57.6541 - val_loss: 41.
1966
Epoch 32/50
1134/1148 ━━━━━━━━━━━━━━━━━━━━ 1s 88ms/step - loss: 56.9344
```

In [*]:

```python
# Evaluate
test_loss = model.evaluate(X_test, y_test)
print(f"Test Loss (MSE): {test_loss}")

# Save model
model.save("melody_generator_model.h5")
```

In [*]:

```python
# Training Loss (MSE) on the training data
train_loss = model.evaluate(X_train, y_train)
print(f"Training Loss (MSE): {train_loss}")

# Calculate test loss (MSE) on the test data
test_loss = model.evaluate(X_test, y_test)
print(f"Test Loss (MSE): {test_loss}")
```

In [*]:

```
from sklearn.metrics import r2_score

# Predict on train and test sets
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

# Calculate R-squared score for training and test sets
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

print(f"Training R^2 Score: {train_r2}")
print(f"Test R^2 Score: {test_r2}")
```

In [*]:

```
import librosa.display
import matplotlib.pyplot as plt

# Function to generate a sequence
def generate_sequence(model, start_sequence, sequence_length):
    generated_sequence = start_sequence.copy()

    for _ in range(sequence_length):
        prediction = model.predict(generated_sequence[-30:].reshape(1, -1,
        generated_sequence = np.vstack([generated_sequence, prediction])

    return generated_sequence
```

In [*]:

```
def mfcc_to_audio(mfcc_sequence, sr=22050):
    S = librosa.feature.inverse.mfcc_to_mel(mfcc_sequence.T, n_mels=128)
    audio = librosa.feature.inverse.mel_to_audio(S, sr=sr)
    return audio
```

In [*]:

```
import IPython.display as ipd

# Load a seed audio file and extract initial MFCCs
y, sr = librosa.load(os.path.join(major_folder, "Major_8.wav"))  # replace
initial_mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc).T[:sequenc

# Generating  melody
generated_mfccs = generate_sequence(model, initial_mfccs, sequence_length=

# Convert generated MFCCs to audio
generated_audio = mfcc_to_audio(generated_mfccs, sr=sr)

# Play generated audio
ipd.Audio(generated_audio, rate=sr)
```

In [*]:
```python
import IPython.display as ipd

# Load a seed audio file and extract initial MFCCs
y, sr = librosa.load(os.path.join(minor_folder, "Minor_8.wav"))  # replace
initial_mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc).T[:sequenc

# Generate a melody
generated_mfccs = generate_sequence(model, initial_mfccs, sequence_length=

# Convert generated MFCCs to audio
generated_audio = mfcc_to_audio(generated_mfccs, sr=sr)

# Play generated audio
ipd.Audio(generated_audio, rate=sr)

```

In [*]:
```python
import librosa.display
import numpy as np
import matplotlib.pyplot as plt

# Function to generate a sequence with added Gaussian noise
def generate_sequence_with_noise(model, start_sequence, sequence_length, r
    generated_sequence = start_sequence.copy()

    for _ in range(sequence_length):
        # Predict the next MFCC frame
        prediction = model.predict(generated_sequence[-30:].reshape(1, -1,

        # Add Gaussian noise to the prediction for variability
        prediction += np.random.normal(0, noise_stddev, prediction.shape)

        # Append the predicted frame to the generated sequence
        generated_sequence = np.vstack([generated_sequence, prediction])

    return generated_sequence
```

In [*]:
```python
# Function to convert MFCC back to audio
def mfcc_to_audio(mfcc_sequence, sr=22050):
    S = librosa.feature.inverse.mfcc_to_mel(mfcc_sequence.T, n_mels=128)
    audio = librosa.feature.inverse.mel_to_audio(S, sr=sr)
    return audio

# Load a seed audio file and extract MFCCs
y, sr = librosa.load(os.path.join(major_folder, "Major_9.wav"))  # replace
initial_mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc).T[:sequenc

```

In [*]:
```python
# Generate the melody with Gaussian noise added
generated_mfccs = generate_sequence_with_noise(model, initial_mfccs, seque

# Convert generated MFCCs to audio
generated_audio = mfcc_to_audio(generated_mfccs, sr=sr)
```

In [*]:

```
# Plot the spectrogram of the generated audio
plt.figure(figsize=(10, 6))
librosa.display.waveshow(generated_audio, sr=sr)
plt.title("Generated Audio Waveform")
plt.show()

plt.figure(figsize = (10,6))
librosa.display.waveshow(y = y, sr = sr)
plt.title("Original Audio Waveform")
plt.show()
```

In [*]:

```
# Play the generated audio
import IPython.display as ipd
ipd.Audio(generated_audio, rate=sr)
```