```
#import google drive
from google.colab import drive
drive.mount('/content/drive')
```

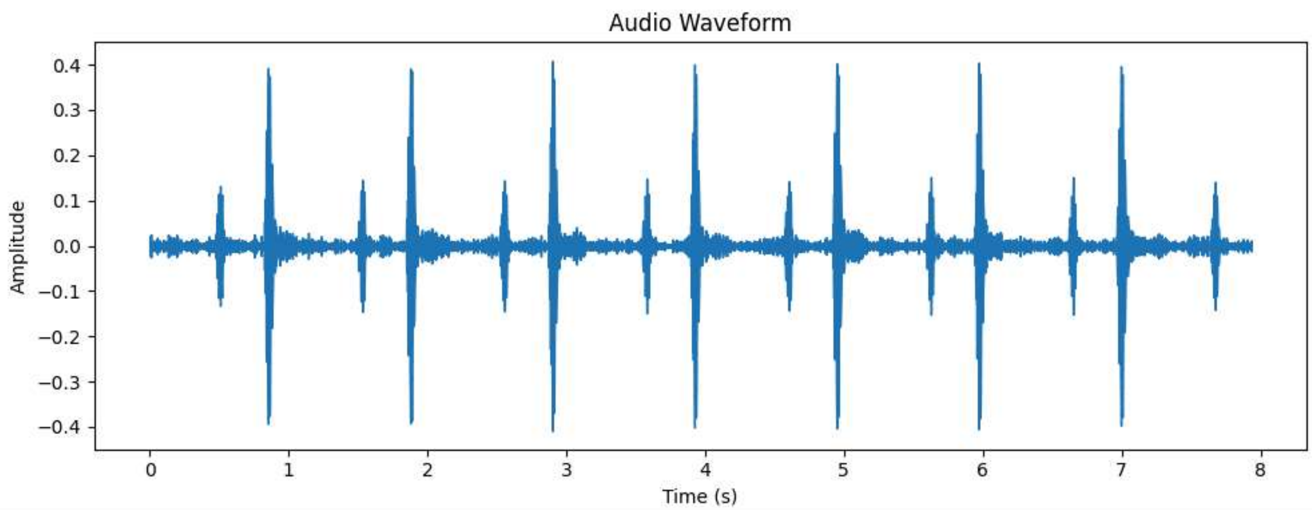
Mounted at /content/drive

```
data_path="/content/drive/MyDrive/set_a"
```

```
import librosa
import librosa.display
import matplotlib.pyplot as plt
import IPython.display as ipd
audio_path = "/content/drive/MyDrive/set_a/normal__201108011114.wav"
y, sr = librosa.load(audio_path, sr=22050)
ipd.Audio(y, rate=sr)
```

0:07 / 0:07

```
plt.figure(figsize=(10, 4))
plt.title("Audio Waveform")
librosa.display.waveshow(y, sr=sr)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.tight_layout()
plt.show()
```




```
import os
import numpy as np
import librosa
import librosa.display
from collections import Counter

# Function to add Gaussian noise
def add_noise(audio, noise_level=0.005):
    noise = np.random.randn(len(audio)) * noise_level
    return audio + noise

# Path to dataset
dataset_path = "/content/drive/MyDrive/set_a"

X = []
y = []
class_counts = Counter()

# Loop through each actor folder
for actor_folder in os.listdir(dataset_path):
    actor_path = os.path.join(dataset_path, actor_folder)
```

```
    if not os.path.isdir(actor_path):
        continue  # Skip non-directory files

    # Loop through each audio file
    for file in os.listdir(actor_path):
        file_path = os.path.join(actor_path, file)

        if not file.endswith(".wav"):
            continue  # Skip non-audio files

        # Load audio
        audio, sr = librosa.load(file_path, sr=22050)

        # Extract MFCC features
        mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=40)
        chroma = librosa.feature.chroma_stft(y=audio, sr=sr)
        spectral_contrast = librosa.feature.spectral_contrast(y=audio, sr=sr)

        # Combine all features
        features = np.concatenate((mfccs.mean(axis=1), chroma.mean(axis=1), spectral_contrast.mean(axis=1)))

        # Extract emotion label (assuming it's the folder name)
        emotion = actor_folder  # Changed to use folder name as label

        # Append to dataset
        X.append(features)
        y.append(emotion)
        class_counts[emotion] += 1

        # If class has fewer samples, augment with noise
        if class_counts[emotion] < 500:  # Adjust this threshold
            audio_noisy = add_noise(audio)
            mfccs_noisy = librosa.feature.mfcc(y=audio_noisy, sr=sr, n_mfcc=40)
            chroma_noisy = librosa.feature.chroma_stft(y=audio_noisy, sr=sr)
            spectral_contrast_noisy = librosa.feature.spectral_contrast(y=audio_noisy, sr=sr)

            features_noisy = np.concatenate((mfccs_noisy.mean(axis=1), chroma_noisy.mean(axis=1), spectral_contrast_noisy.mean(axis=1)))

            X.append(features_noisy)
            y.append(emotion)
            class_counts[emotion] += 1  # Update count after augmentation

print("Balanced Class Distribution:", class_counts)
```

```
Balanced Class Distribution: Counter()
```

```
import os
import numpy as np
import librosa
import librosa.display
from collections import Counter

# Function to add Gaussian noise
def add_noise(audio, noise_level=0.005):
    noise = np.random.randn(len(audio)) * noise_level
    return audio + noise

# Path to dataset
dataset_path = "/content/drive/MyDrive/set_a"

X = []
y = []
class_counts = Counter()

# Loop through each actor folder
for actor_folder in os.listdir(dataset_path):
    actor_path = os.path.join(dataset_path, actor_folder)

    if not os.path.isdir(actor_path):
        continue  # Skip non-directory files

    # Loop through each audio file
    for file in os.listdir(actor_path):
        file_path = os.path.join(actor_path, file)
```

```
        if not file.endswith(".wav"):
            continue  # Skip non-audio files

        # Load audio
        audio, sr = librosa.load(file_path, sr=22050)

        # Extract MFCC features
        mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=40)
        chroma = librosa.feature.chroma_stft(y=audio, sr=sr)
        spectral_contrast = librosa.feature.spectral_contrast(y=audio, sr=sr)

        # Combine all features
        features = np.concatenate((mfccs.mean(axis=1), chroma.mean(axis=1), spectral_contrast.mean(axis=1)))

        # Extract emotion label (assuming it's the folder name)
        emotion = actor_folder  # Changed to use folder name as label

        # Append to dataset
        X.append(features)
        y.append(emotion)
        class_counts[emotion] += 1

        # If class has fewer samples, augment with noise
        if class_counts[emotion] < 500:  # Adjust this threshold
            audio_noisy = add_noise(audio)
            mfccs_noisy = librosa.feature.mfcc(y=audio_noisy, sr=sr, n_mfcc=40)
            chroma_noisy = librosa.feature.chroma_stft(y=audio_noisy, sr=sr)
            spectral_contrast_noisy = librosa.feature.spectral_contrast(y=audio_noisy, sr=sr)

            features_noisy = np.concatenate((mfccs_noisy.mean(axis=1), chroma_noisy.mean(axis=1), spectral_contrast_noisy.mean(axis=1)))

            X.append(features_noisy)
            y.append(emotion)
            class_counts[emotion] += 1  # Update count after augmentation

print("Balanced Class Distribution:", class_counts)
```

```
Balanced Class Distribution: Counter()
```

```
import os
import librosa
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical

# Path to audio files
audio_dir = "/content/drive/MyDrive/set_a"

# Parameters
max_len = 130  # max number of MFCC frames
n_mfcc = 13    # number of MFCCs per frame

# Storage
X = []
y = []

# Load audio files and extract MFCCs
for file in os.listdir(audio_dir):
    if file.endswith(".wav"):
        file_path = os.path.join(audio_dir, file)
        label = file.split("_")[0]  # Assume label is part of filename (e.g., "dog_bark.wav")

        audio, sr = librosa.load(file_path, sr=None)
        mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc)
        mfcc = mfcc.T  # Transpose to shape (time, features)

        X.append(mfcc)
        y.append(label)

# Pad sequences
X = pad_sequences(X, maxlen=max_len, padding='post', dtype='float32')

# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

```
y_categorical = to_categorical(y_encoded)

print("X shape:", X.shape)  # (samples, timesteps, features)
print("y shape:", y_categorical.shape)
```

```
X shape: (176, 130, 13)
y shape: (176, 5)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Masking

model = Sequential()
model.add(Masking(mask_value=0., input_shape=(max_len, n_mfcc)))
model.add(LSTM(128))
model.add(Dense(len(np.unique(y_encoded)), activation='softmax'))

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

model.fit(X, y_categorical, epochs=20, batch_size=16, validation_split=0.2)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/masking.py:47: UserWarning: Do not pass an `input_shape`/`input_dim` argum
  super().__init__(**kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Param #
masking (Masking)	(None, 130, 13)	0
lstm (LSTM)	(None, 128)	72,704
dense (Dense)	(None, 5)	645

```
 Total params: 73,349 (286.52 KB)
 Trainable params: 73,349 (286.52 KB)
 Non-trainable params: 0 (0.00 B)
Epoch 1/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 4s 75ms/step - accuracy: 0.2575 - loss: 1.6355 - val_accuracy: 0.0833 - val_loss: 2.9559
Epoch 2/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.4261 - loss: 1.3385 - val_accuracy: 0.0278 - val_loss: 3.2808
Epoch 3/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.5126 - loss: 1.1919 - val_accuracy: 0.0278 - val_loss: 3.5535
Epoch 4/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.5733 - loss: 1.1303 - val_accuracy: 0.0278 - val_loss: 3.7166
Epoch 5/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step - accuracy: 0.6669 - loss: 1.0370 - val_accuracy: 0.0556 - val_loss: 3.8904
Epoch 6/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.6194 - loss: 1.0390 - val_accuracy: 0.1111 - val_loss: 3.9853
Epoch 7/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step - accuracy: 0.6173 - loss: 0.9457 - val_accuracy: 0.1111 - val_loss: 4.1792
Epoch 8/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.6275 - loss: 0.9277 - val_accuracy: 0.1111 - val_loss: 4.3935
Epoch 9/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.5947 - loss: 0.9406 - val_accuracy: 0.1111 - val_loss: 4.4440
Epoch 10/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.6002 - loss: 0.8843 - val_accuracy: 0.1111 - val_loss: 4.6229
Epoch 11/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.6071 - loss: 0.8798 - val_accuracy: 0.0833 - val_loss: 4.7571
Epoch 12/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.6726 - loss: 0.8745 - val_accuracy: 0.1111 - val_loss: 4.9081
Epoch 13/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.6776 - loss: 0.7801 - val_accuracy: 0.1111 - val_loss: 4.9754
Epoch 14/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.6370 - loss: 0.8142 - val_accuracy: 0.1111 - val_loss: 5.0941
Epoch 15/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - accuracy: 0.6720 - loss: 0.7783 - val_accuracy: 0.1111 - val_loss: 5.1295
Epoch 16/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.6624 - loss: 0.7940 - val_accuracy: 0.1111 - val_loss: 5.1956
Epoch 17/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.6535 - loss: 0.7867 - val_accuracy: 0.1111 - val_loss: 5.2794
Epoch 18/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step - accuracy: 0.6768 - loss: 0.7226 - val_accuracy: 0.1111 - val_loss: 5.3529
Epoch 19/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - accuracy: 0.6513 - loss: 0.7118 - val_accuracy: 0.1111 - val_loss: 5.4333
Epoch 20/20
9/9 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.7273 - loss: 0.6889 - val_accuracy: 0.1111 - val_loss: 5.4174
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, BatchNormalization
from sklearn.model_selection import train_test_split # Import train_test_split

# Assuming X and y_categorical are already defined from previous cells

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_categorical, test_size=0.2, random_state=42) # Split the data

# Define the LSTM model
model = Sequential([
    LSTM(128, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])), # Use X_train.shape
    BatchNormalization(),
    Dropout(0.3),

    LSTM(128, return_sequences=True),  # First additional LSTM layer
    BatchNormalization(),
    Dropout(0.3),

    LSTM(64, return_sequences=True),  # Second additional LSTM layer
    BatchNormalization(),
    Dropout(0.3),

    LSTM(64),  # Final LSTM layer
    BatchNormalization(),
    Dropout(0.3),

    Dense(32, activation='relu'),
    Dropout(0.3),

    Dense(8, activation='softmax')  # Output layer (8 classes for emotions)
])

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

# Print the model summary
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(**kwargs)
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 130, 128)	72,704
batch_normalization (BatchNormalization)	(None, 130, 128)	512
dropout (Dropout)	(None, 130, 128)	0
lstm_2 (LSTM)	(None, 130, 128)	131,584
batch_normalization_1 (BatchNormalization)	(None, 130, 128)	512
dropout_1 (Dropout)	(None, 130, 128)	0
lstm_3 (LSTM)	(None, 130, 64)	49,408
batch_normalization_2 (BatchNormalization)	(None, 130, 64)	256
dropout_2 (Dropout)	(None, 130, 64)	0
lstm_4 (LSTM)	(None, 64)	33,024
batch_normalization_3 (BatchNormalization)	(None, 64)	256
dropout_3 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
dropout_4 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 8)	264

Total params: 290,600 (1.11 MB)
Trainable params: 289,832 (1.11 MB)

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, BatchNormalization
from sklearn.model_selection import train_test_split

# Assuming X and y_categorical are already defined from previous cells

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_categorical, test_size=0.2, random_state=42) # Split the data

# Define the LSTM model
model = Sequential([
    LSTM(128, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])), # Use X_train.shape
    BatchNormalization(),
    Dropout(0.3),

    LSTM(128, return_sequences=True),  # First additional LSTM layer
    BatchNormalization(),
    Dropout(0.3),

    LSTM(64, return_sequences=True),  # Second additional LSTM layer
    BatchNormalization(),
    Dropout(0.3),

    LSTM(64),  # Final LSTM layer
    BatchNormalization(),
    Dropout(0.3),

    Dense(32, activation='relu'),
    Dropout(0.3),

    Dense(y_train.shape[1], activation='softmax')  # Output layer with the correct number of classes
])

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

```
# Print the model summary
model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
lstm_5 (LSTM)	(None, 130, 128)	72,704
batch_normalization_4 (BatchNormalization)	(None, 130, 128)	512
dropout_5 (Dropout)	(None, 130, 128)	0
lstm_6 (LSTM)	(None, 130, 128)	131,584
batch_normalization_5 (BatchNormalization)	(None, 130, 128)	512
dropout_6 (Dropout)	(None, 130, 128)	0
lstm_7 (LSTM)	(None, 130, 64)	49,408
batch_normalization_6 (BatchNormalization)	(None, 130, 64)	256
dropout_7 (Dropout)	(None, 130, 64)	0
lstm_8 (LSTM)	(None, 64)	33,024
batch_normalization_7 (BatchNormalization)	(None, 64)	256
dropout_8 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 32)	2,080
dropout_9 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 5)	165

Total params: 290,501 (1.11 MB)
Trainable params: 289,733 (1.11 MB)

```
history = model.fit(X_train, y_train, epochs=15, batch_size=32, validation_data=(X_test, y_test))
```

```
Epoch 1/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 216ms/step - accuracy: 0.2183 - loss: 2.2647 - val_accuracy: 0.3889 - val_loss: 1.5717
Epoch 2/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step - accuracy: 0.3228 - loss: 1.8898 - val_accuracy: 0.4167 - val_loss: 1.5541
Epoch 3/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - accuracy: 0.4330 - loss: 1.5583 - val_accuracy: 0.3889 - val_loss: 1.5432
Epoch 4/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - accuracy: 0.3991 - loss: 1.6306 - val_accuracy: 0.4167 - val_loss: 1.5408
Epoch 5/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.4575 - loss: 1.4602 - val_accuracy: 0.3333 - val_loss: 1.5487
Epoch 6/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.4239 - loss: 1.3989 - val_accuracy: 0.4167 - val_loss: 1.5505
Epoch 7/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - accuracy: 0.4865 - loss: 1.2672 - val_accuracy: 0.3889 - val_loss: 1.5399
Epoch 8/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.5694 - loss: 1.1067 - val_accuracy: 0.3889 - val_loss: 1.5158
Epoch 9/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.5488 - loss: 1.2811 - val_accuracy: 0.3889 - val_loss: 1.5112
Epoch 10/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - accuracy: 0.5659 - loss: 1.1616 - val_accuracy: 0.3889 - val_loss: 1.5052
Epoch 11/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - accuracy: 0.5503 - loss: 1.2332 - val_accuracy: 0.4167 - val_loss: 1.4838
Epoch 12/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - accuracy: 0.5592 - loss: 1.0905 - val_accuracy: 0.4444 - val_loss: 1.4682
Epoch 13/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - accuracy: 0.6000 - loss: 1.0102 - val_accuracy: 0.4722 - val_loss: 1.4526
Epoch 14/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 56ms/step - accuracy: 0.6952 - loss: 0.8447 - val_accuracy: 0.4444 - val_loss: 1.4401
Epoch 15/15
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step - accuracy: 0.6364 - loss: 0.9162 - val_accuracy: 0.4167 - val_loss: 1.4059
```

```
#Example of using RMSprop optimizer with adjusted learning rate:
from tensorflow.keras.optimizers import RMSprop
```

```
optimizer = RMSprop(learning_rate=0.0005)  # Adjust learning rate as needed
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])


# Example of Time Stretching
from librosa import effects

audio_stretched = effects.time_stretch(audio, rate=1.2)  # Stretch by 20%


# Evaluate on test data
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 35ms/step - accuracy: 0.4028 - loss: 1.4189
Test Accuracy: 41.67%
```

```
import os
import librosa
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, BatchNormalization
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Path to audio files
audio_dir = "/content/drive/MyDrive/set_a"

# Parameters
max_len = 130  # max number of MFCC frames
n_mfcc = 13    # number of MFCCs per frame

# Storage
X = []
y = []

# Load audio files and extract MFCCs
for file in os.listdir(audio_dir):
    if file.endswith(".wav"):
        file_path = os.path.join(audio_dir, file)
        label = file.split("_")[0]  # Assume label is part of filename (e.g., "dog_bark.wav")

        audio, sr = librosa.load(file_path, sr=None)
        mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc)
        mfcc = mfcc.T  # Transpose to shape (time, features)

        X.append(mfcc)
        y.append(label)

# Pad sequences
X = pad_sequences(X, maxlen=max_len, padding='post', dtype='float32')

# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_categorical = to_categorical(y_encoded)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_categorical, test_size=0.2, random_state=42)

# Define the LSTM model (adjust as needed)
model = Sequential([
    LSTM(128, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])),
    BatchNormalization(),
    Dropout(0.3),
    LSTM(128, return_sequences=True),
    BatchNormalization(),
    Dropout(0.3),
    LSTM(64, return_sequences=True),
    BatchNormalization(),
    Dropout(0.3),
    LSTM(64)
```

```
    LSTM(64),
    BatchNormalization(),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.3),
    Dense(y_train.shape[1], activation='softmax')
])

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

# Train the model (adjust epochs and batch size as needed)
# model.fit(X_train, y_train, epochs=15, batch_size=32, validation_data=(X_test, y_test))

# Predict on the test data
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)  # Convert predictions to class labels
y_true_classes = np.argmax(y_test, axis=1)  # Convert true labels to class labels

# Generate confusion matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)

# Display confusion matrix using seaborn heatmap
plt.figure(figsize=(6, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument
  super().__init__(**kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 372ms/step
```