IIT-H: Reasearch Teaser

Batch 4

Digit Automated Speech Recognition

Mohammad Afran

Navaneeth Vishnu K

Sajin Satheesh Kumar

04/12/2024

Project Report

Index

Abstract

This project focuses on developing an automated system for recognizing spoken digits, utilizing Mel-Frequency Cepstral Coefficients (MFCCs) to extract key audio features and Long Short-Term Memory (LSTM) networks to analyze temporal patterns. The system, trained on the Free Spoken Digit Dataset (FSDD), effectively classifies digits with promising accuracy. Future directions include expanding the system’s vocabulary, enhancing its ability to handle noisy environments, and implementing real-time recognition capabilities. These improvements could enable practical applications in voice-controlled systems, assistive technologies, and various industries such as telecommunications and healthcare, education, and smart devices, revolutionizing human-computer interaction and accessibility.

Introduction

Speech recognition is a transformative technology reshaping human-computer interaction by enabling intuitive, hands-free communication. This project explores the design and development of an automated system for spoken digit recognition, leveraging MelFrequency Cepstral Coefficients (MFCCs) to extract crucial audio features and Long Short-Term Memory (LSTM) networks to model temporal dependencies. By focusing on digit classification, the study highlights how the integration of traditional signal processing methods with cutting-edge deep learning techniques can achieve accurate and reliable results. The system, built using the Free Spoken Digit Dataset (FSDD), offers valuable insights into the potential of speech recognition technology, opening doors to applications in voice assistants, accessibility tools, and emerging domains such as smart devices, healthcare, and personalized education.

Problem Statement

In today's rapidly evolving technological landscape, efficient and accurate speech recognition systems are essential for enabling seamless human-computer interaction. While significant advancements have been made in recognizing complex speech patterns, there remains a gap in the development of systems that can reliably classify spoken digits with precision and adaptability across varying conditions. The challenge lies in designing a system that can extract and analyze the key features of audio signals while accounting for temporal dependencies in speech data. Addressing these issues is critical for advancing applications in voiceactivated technologies, assistive tools, and various industries requiring intuitive and accessible solutions. This project aims to tackle these challenges by developing an automated spoken digit recognition system using MFCCs for feature extraction and LSTM networks for temporal pattern analysis, providing a foundation for practical and innovative speech recognition applications.

Methodology

The development of an automated spoken digit recognition system involves distinct phases: data collection, preprocessing, feature extraction, model development, and evaluation. This structured approach ensures reliability and adaptability.

Data Collection

Dataset: Free Spoken Digit Dataset (FSDD), containing WAV files at 8 kHz from diverse speakers and environments.

Preprocessing

Objective: Standardize audio for consistent feature extraction.

Steps: Audio loading via Librosa, noise reduction, and amplitude normalization.

Feature Extraction

MFCCs: Extracted to represent speech features.

13 coefficients per frame using a short time window.

Mean values calculated for fixed-size input.

Model Architecture

LSTM Network: Captures temporal patterns.

Input layer for MFCCs.

LSTM layers, dense layers, and a softmax output for digit classification.

Training

Dataset Split: 70% training, 20% validation, 10% testing.

Loss/Optimizer: Categorical Cross-Entropy with Adam optimizer.

Hyperparameters: 20 epochs, batch size of 32, and tuned learning rate.

Evaluation

Metrics: Accuracy, loss, and confusion matrix for performance analysis.

Deployment

Pipeline: Preprocessed audio is converted to MFCCs, passed to the model, and outputs predictions in real-time.

Platform: Python-based, with potential mobile/web integration.

Enhancements

Experiment with features like spectrograms, additional LSTM layers, and data augmentation to improve performance and robustness.

This modular methodology ensures a scalable, efficient spoken digit recognition system with applications in various speech technologies.

Specification

The implementation of the automated spoken digit recognition system utilized the following tools and resources:

Development Platform

Google Colab: A cloud-based environment offering GPU support for efficient model training and accessibility.

Programming Language

Python 3.x: Selected for its flexibility and compatibility with data science and machine learning tasks.

Libraries and Frameworks

TensorFlow: Used to design and train the LSTM model.

Librosa: For audio processing and feature extraction (e.g., MFCCs).

NumPy: Enabled efficient numerical operations.

Scikit-learn: Assisted with preprocessing, encoding, and evaluation.

Dataset

Free Spoken Digit Dataset (FSDD): Sourced directly from GitHub for ease of access and reproducibility.

https://github.com/Jakobovski/free-spoken-digit-dataset.git

Visualization Tools

Matplotlib: For visualizing data and results, including confusion matrices.

Librosa.display: Provided visual representations of audio features like waveforms and spectrograms.

Hardware

Google Colab's cloud-based runtime powered by Google Compute Engine ensured faster processing and model training.

Additional Utilities

- Tools like OS for file management and LabelEncoder from Scikit-learn for preprocessing tasks.

This streamlined setup ensured efficient development and deployment of the spoken digit recognition system.

Program & Output

Original file is located at:
https://colab.research.google.com/drive/15Pk-g1rJSMIf-dfKMcoyw9Z7lALJqpip?usp=sharing

```
!pip install tensorflow
!pip install librosa
!pip install numpy
!pip install scikit-learn
```

```
import os
import librosa
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, Flatten
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
```

```
!git clone https://github.com/Jakobovski/free-spoken-digit-dataset.git
```

```python
import librosa.display  # Import for visualizing MFCCs

# Path to the recordings directory
audio_path = '/content/free-spoken-digit-dataset/recordings/'

# Initialize data and labels lists
X = []
y = []

# Extract MFCC features
for file in os.listdir(audio_path):
    if file.endswith(".wav"):
        # Load the audio file
        audio_file = os.path.join(audio_path, file)
        signal, sr = librosa.load(audio_file, sr=None)

        # Extract MFCCs (Mel-Frequency Cepstral Coefficients)
        # The function is called with the expected argument y=signal
        mfcc = librosa.feature.mfcc(y=signal, sr=sr, n_mfcc=13)
        mfcc = np.mean(mfcc.T, axis=0)  # Average over time (axis=0)

        X.append(mfcc)

        # Label extraction: The label is the digit in the filename
        label = file.split("_")[0]
        y.append(label)

# Convert data and labels to numpy arrays
X = np.array(X)
y = np.array(y)

# Print the shape of X and y
print(X.shape, y.shape)

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
print("Encoded labels:", label_encoder.classes_)

print(f"Labels count: {np.unique(y, return_counts=True)}")
```

```
(3000, 13) (3000,)
Encoded labels: ['0' '1' '2' '3' '4' '5' '6' '7' '8' '9']
Labels count: (array(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'], dtype='<U1'), array([300, 300, 300, 300, 300, 300, 300, 300, 300, 300]))
```

```
# Encode labels (0-9)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Print some encoded labels to verify
print(y[:10], y_encoded[:10])
```

```
['8' '1' '4' '6' '1' '6' '4' '4' '3' '9'] [8 1 4 6 1 6 4 4 3 9]
```

```
X_train, X_val, y_train, y_val = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

# Print the shapes of the split data
print(X_train.shape, X_val.shape)
```

```
(2400, 13) (600, 13)
```

```
# Define the model architecture
model = Sequential()

# Add an LSTM layer
model.add(LSTM(128, input_shape=(X_train.shape[1], 1), return_sequences=True))
model.add(Dropout(0.3))

# Add another LSTM layer
model.add(LSTM(128))
model.add(Dropout(0.3))

# Add a Dense layer for classification
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))

# Output layer with softmax activation (10 classes: digits 0-9)
model.add(Dense(10, activation='softmax'))

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

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

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 13, 128)	66,560
dropout (Dropout)	(None, 13, 128)	0
lstm_1 (LSTM)	(None, 128)	131,584
dropout_1 (Dropout)	(None, 128)	0
dense (Dense)	(None, 64)	8,256
dropout_2 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 10)	650

Total params: 207,050 (808.79 KB)
Trainable params: 207,050 (808.79 KB)
Non-trainable params: 0 (0.00 B)

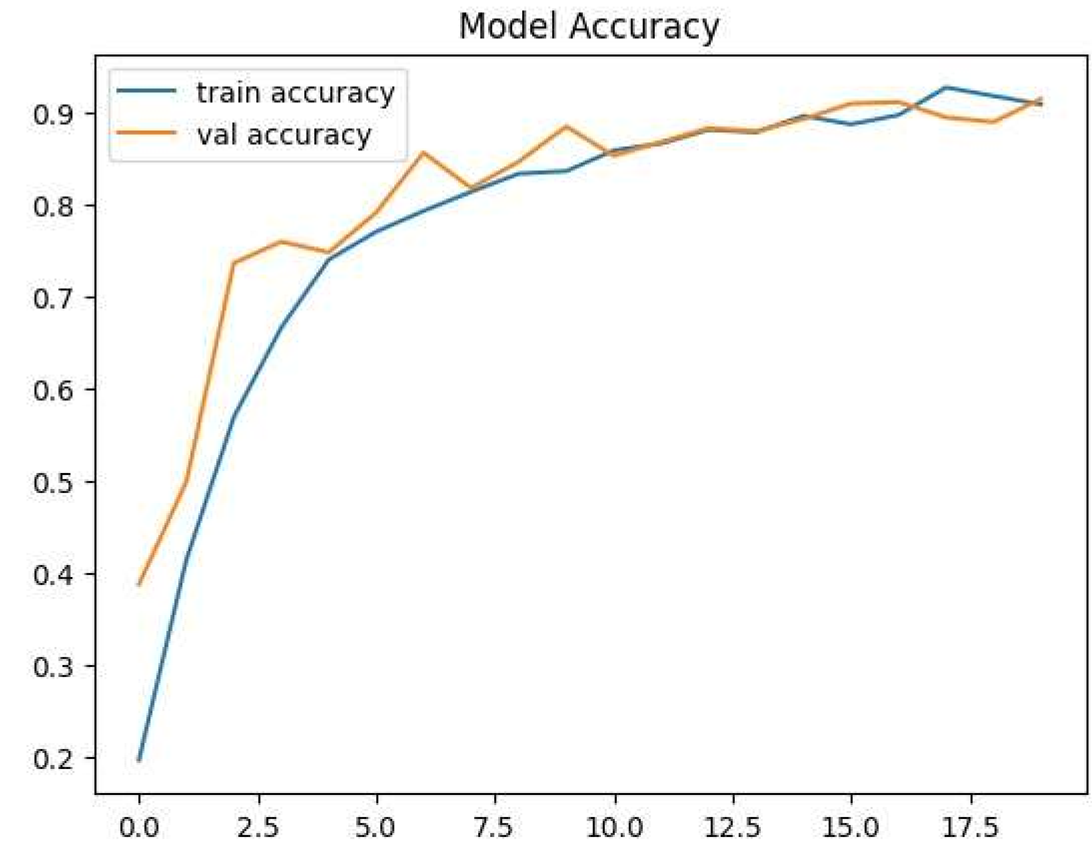
```python
# Reshape the data to fit LSTM input requirements
X_train = X_train[..., np.newaxis]
X_val = X_val[..., np.newaxis]

# Print reshaped data shapes
print(X_train.shape, X_val.shape)
```

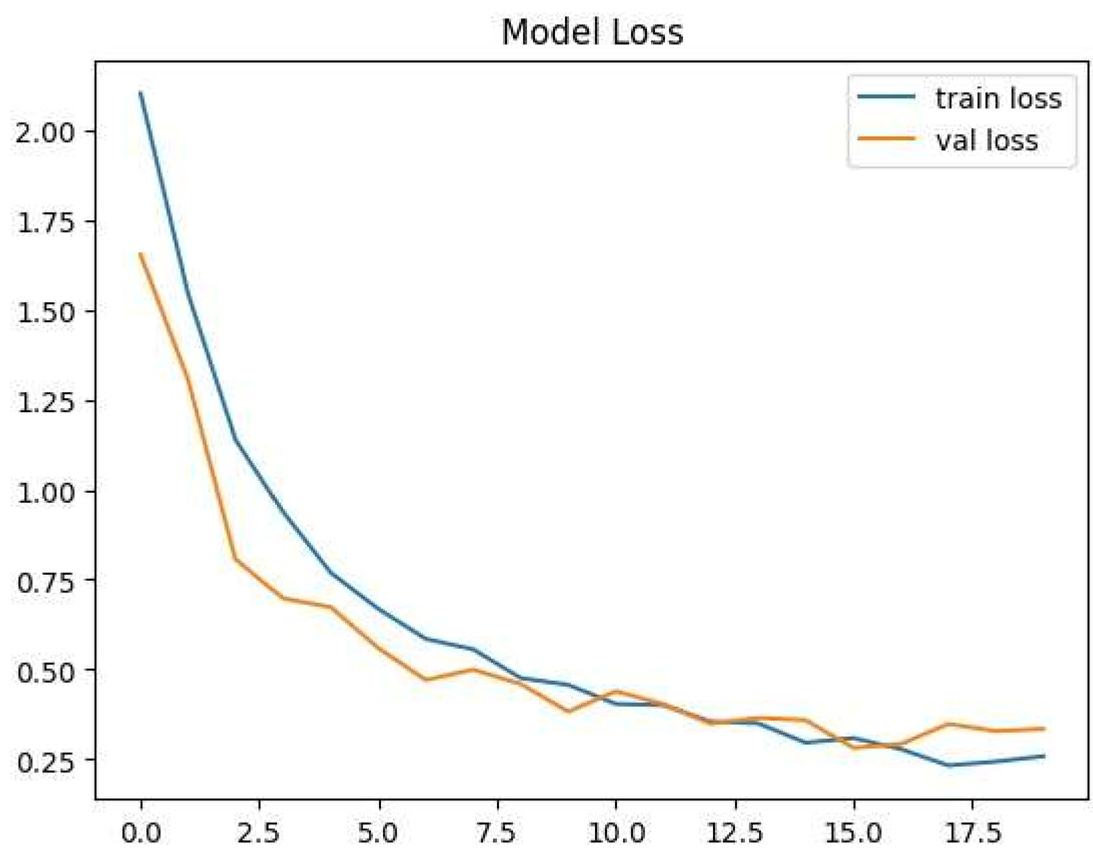
```
(2400, 13, 1) (600, 13, 1)
```

```python
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_val, y_val))

# Plot the training & validation accuracy
plt.plot(history.history['accuracy'], label='train accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.legend()
plt.title('Model Accuracy')
plt.show()

# Plot the training & validation loss
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.legend()
plt.title('Model Loss')
plt.show()
```

```
Epoch 1/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 8s 50ms/step - accuracy: 0.1427 - loss: 2.2357 - val_accuracy: 0.3883 - val_loss: 1.6531
Epoch 2/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 4s 40ms/step - accuracy: 0.3706 - loss: 1.6416 - val_accuracy: 0.5000 - val_loss: 1.3060
Epoch 3/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 4s 54ms/step - accuracy: 0.5379 - loss: 1.2319 - val_accuracy: 0.7367 - val_loss: 0.8069
Epoch 4/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 66ms/step - accuracy: 0.6529 - loss: 0.9598 - val_accuracy: 0.7600 - val_loss: 0.6975
Epoch 5/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - accuracy: 0.7450 - loss: 0.7750 - val_accuracy: 0.7483 - val_loss: 0.6734
Epoch 6/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 3s 42ms/step - accuracy: 0.7652 - loss: 0.6689 - val_accuracy: 0.7917 - val_loss: 0.5595
Epoch 7/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 63ms/step - accuracy: 0.7832 - loss: 0.6158 - val_accuracy: 0.8567 - val_loss: 0.4708
Epoch 8/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 4s 42ms/step - accuracy: 0.8317 - loss: 0.5206 - val_accuracy: 0.8183 - val_loss: 0.4996
Epoch 9/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 43ms/step - accuracy: 0.8288 - loss: 0.4837 - val_accuracy: 0.8467 - val_loss: 0.4597
Epoch 10/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 62ms/step - accuracy: 0.8377 - loss: 0.4718 - val_accuracy: 0.8850 - val_loss: 0.3827
Epoch 11/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 4s 42ms/step - accuracy: 0.8608 - loss: 0.4073 - val_accuracy: 0.8533 - val_loss: 0.4392
Epoch 12/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 5s 45ms/step - accuracy: 0.8698 - loss: 0.3986 - val_accuracy: 0.8683 - val_loss: 0.4037
Epoch 13/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 7s 69ms/step - accuracy: 0.8681 - loss: 0.3856 - val_accuracy: 0.8833 - val_loss: 0.3491
Epoch 14/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 8s 44ms/step - accuracy: 0.8783 - loss: 0.3574 - val_accuracy: 0.8800 - val_loss: 0.3651
Epoch 15/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 6s 55ms/step - accuracy: 0.8921 - loss: 0.3014 - val_accuracy: 0.8933 - val_loss: 0.3592
Epoch 16/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 4s 46ms/step - accuracy: 0.8900 - loss: 0.2856 - val_accuracy: 0.9100 - val_loss: 0.2817
```

```
Epoch 17/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 6s 62ms/step - accuracy: 0.9041 - loss: 0.2705 - val_accuracy: 0.9117 - val_loss: 0.2925
Epoch 18/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 4s 53ms/step - accuracy: 0.9255 - loss: 0.2379 - val_accuracy: 0.8950 - val_loss: 0.3485
Epoch 19/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 4s 43ms/step - accuracy: 0.9221 - loss: 0.2285 - val_accuracy: 0.8900 - val_loss: 0.3286
Epoch 20/20
75/75 ━━━━━━━━━━━━━━━━━━━━ 7s 66ms/step - accuracy: 0.9139 - loss: 0.2599 - val_accuracy: 0.9150 - val_loss: 0.3346
```


Model Accuracy
train accuracy
val accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Model Loss
train loss
val loss
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

```
val_loss, val_accuracy = model.evaluate(X_val, y_val)
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 26ms/step - accuracy: 0.9103 - loss: 0.3492
Validation Accuracy: 91.50%
```

```
from google.colab import files

# Step 1: Upload the audio file
uploaded = files.upload()

# Get the uploaded file name
audio_path = list(uploaded.keys())[0]
print(f"Uploaded file: {audio_path}")

# Step 2: Load and preprocess the uploaded audio file
# Load the audio file using librosa
signal, sr = librosa.load(audio_path, sr=None)

# Extract MFCC features (Mel-Frequency Cepstral Coefficients)
# Use keyword arguments for 'y' and 'sr'
mfcc = librosa.feature.mfcc(y=signal, sr=sr, n_mfcc=13)
mfcc = np.mean(mfcc.T, axis=0)  # Average over time (axis=0)

# Reshape the MFCC array for LSTM input (1 sample, 13 features, 1 channel)
sample = mfcc.reshape(1, -1, 1)

# Print the shape of the sample to verify
print(f"Sample shape: {sample.shape}")

# Step 3: Predict using the trained model
# Assuming you already have a trained model and label_encoder from previous training steps

# Predict the digit
prediction = model.predict(sample)

# Get the predicted digit by finding the class with the highest probability
predicted_digit = label_encoder.inverse_transform([np.argmax(prediction)])

# Step 4: Print the predicted digit
print(f"Predicted digit: {predicted_digit[0]}")
```

```
Choose Files 0_george_7.wav
• 0_george_7.wav(audio/wav) - 10806 bytes, last modified: 12/6/2024 - 100% done
Saving 0_george_7.wav to 0_george_7 (1).wav
Uploaded file: 0_george_7 (1).wav
Sample shape: (1, 13, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 316ms/step
Predicted digit: 0
```

```
Choose Files 1_jackson_46.wav
• 1_jackson_46.wav(audio/wav) - 8576 bytes, last modified: 12/6/2024 - 100% done
Saving 1_jackson_46.wav to 1_jackson_46.wav
Uploaded file: 1_jackson_46.wav
Sample shape: (1, 13, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
Predicted digit: 1
```

```
Choose Files 2_jackson_37.wav
• 2_jackson_37.wav(audio/wav) - 6508 bytes, last modified: 12/6/2024 - 100% done
Saving 2_jackson_37.wav to 2_jackson_37.wav
Uploaded file: 2_jackson_37.wav
Sample shape: (1, 13, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
Predicted digit: 2
```

```
Choose Files 3_lucas_8.wav
• 3_lucas_8.wav(audio/wav) - 9532 bytes, last modified: 12/6/2024 - 100% done
Saving 3_lucas_8.wav to 3_lucas_8.wav
Uploaded file: 3_lucas_8.wav
Sample shape: (1, 13, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
Predicted digit: 3
```

```
Choose Files 4_jackson_17.wav
• 4_jackson_17.wav(audio/wav) - 6490 bytes, last modified: 12/6/2024 - 100% done
Saving 4_jackson_17.wav to 4_jackson_17.wav
Uploaded file: 4_jackson_17.wav
Sample shape: (1, 13, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step
Predicted digit: 4
```

Choose Files 5_lucas_29.wav

- **5_lucas_29.wav**(audio/wav) - 8742 bytes, last modified: 12/6/2024 - 100% done

```
Saving 5_lucas_29.wav to 5_lucas_29.wav
Uploaded file: 5_lucas_29.wav
Sample shape: (1, 13, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
Predicted digit: 5
```

Choose Files 6_lucas_19.wav

- **6_lucas_19.wav**(audio/wav) - 9436 bytes, last modified: 12/6/2024 - 100% done

```
Saving 6_lucas_19.wav to 6_lucas_19.wav
Uploaded file: 6_lucas_19.wav
Sample shape: (1, 13, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step
Predicted digit: 6
```

Choose Files 7_george_38.wav

- **7_george_38.wav**(audio/wav) - 8216 bytes, last modified: 12/6/2024 - 100% done

```
Saving 7_george_38.wav to 7_george_38.wav
Uploaded file: 7_george_38.wav
Sample shape: (1, 13, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
Predicted digit: 7
```

Choose Files 8_lucas_21.wav

- **8_lucas_21.wav**(audio/wav) - 11276 bytes, last modified: 12/6/2024 - 100% done

```
Saving 8_lucas_21.wav to 8_lucas_21.wav
Uploaded file: 8_lucas_21.wav
Sample shape: (1, 13, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step
Predicted digit: 8
```

Choose Files 9_george_27.wav

- **9_george_27.wav**(audio/wav) - 7424 bytes, last modified: 12/6/2024 - 100% done

```
Saving 9_george_27.wav to 9_george_27.wav
Uploaded file: 9_george_27.wav
Sample shape: (1, 13, 1)
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step
Predicted digit: 9
```

Conclusion

This project aims to develop an automated system for identifying spoken digits, utilizing Mel-Frequency Cepstral Coefficients (MFCCs) to extract essential audio features and Long Short-Term Memory (LSTM) networks to analyze temporal patterns. The system, trained on the Free Spoken Digit Dataset (FSDD), demonstrates promising accuracy in digit classification. Future enhancements include improving performance in noisy environments, and integrating real-time recognition capabilities. These advancements could enable practical applications in voice-activated systems, assistive technologies, and industries such as telecommunications. Moreover, integrating such systems into everyday devices has the potential to enhance accessibility, streamline workflows, and transform personalized user experiences. As research in this area progresses, it holds significant promise to revolutionize industries and push the boundaries of technology-driven communication.