Speech Recognition System for Digits from 0 to 9

Abdelilah Bajjou
La Faculté des Sciences Dhar El Mahraz – FSDM
December 1, 2024

Abstract

The goal of this project is to develop a speech recognition system that identifies spoken digits (0–9) from audio recordings. The system uses a Long Short-Term Memory (LSTM) neural network to classify the digits based on features extracted from the audio files. The audio data is preprocessed using Mel-Frequency Cepstral Coefficients (MFCCs), which are commonly used features in speech recognition tasks. A dataset containing audio recordings of both male and female speakers was used to train and test the model. The model was trained to recognize digits with high accuracy, achieving a final accuracy of 96%. Additionally, the system includes a graphical user interface (GUI) that allows users to predict the digit spoken by simply recording their voice or uploading an audio file. The GUI makes the system user-friendly, enabling real-time interaction with the model.

Contents

Chapter 1

Introduction

1.1 Context and Motivation

Speech recognition is a crucial area of research in natural language processing (NLP) and artificial intelligence (AI). It involves converting spoken language into text, enabling machines to understand and respond to human speech. Applications of speech recognition span across various industries, including voice-controlled assistants (e.g., Siri, Alexa), accessibility tools for the disabled, and automated transcription services. With the advancement of deep learning techniques, speech recognition systems have achieved remarkable accuracy, particularly in controlled environments.

1.2 Problem Statement

The focus of this project is to develop a speech recognition system capable of identifying spoken digits from 0 to 9. These digits are often used as inputs for various systems, such as automated phone services and banking systems. The challenge lies in accurately recognizing spoken digits from audio recordings that may come from different speakers (both male and female), each with varying speech patterns and environmental conditions. This task is made more complex by the inherent variability in speech, such as differences in pronunciation, speed, and background noise.

1.3 Objectives

The primary objective of this project is to design and implement a system that can recognize digits (0-9) spoken by users, regardless of their gender. This system uses a Long Short-Term Memory (LSTM) neural network for classification. The key

objectives are as follows:

- Preprocess audio data using Mel-Frequency Cepstral Coefficients (MFCC) for feature extraction.
- Train an LSTM model to recognize digits based on the extracted features.
- Create a graphical user interface (GUI) that allows users to either record their voice or upload an audio file for digit prediction.

1.4 Approach

To achieve the objectives, the project leverages a combination of techniques from deep learning and audio signal processing:

- The audio data is preprocessed using MFCCs, which capture the essential features of the speech signal.
- The LSTM neural network is employed due to its ability to learn sequential patterns, making it well-suited for speech recognition tasks.
- The system is integrated with a GUI, allowing real-time interaction where users can record their voice or upload audio files for digit prediction.

This approach ensures that the system is both accurate and user-friendly, making it a valuable tool for speech recognition in various applications.

Chapter 2

Methodology

2.1 Data Collection

The dataset used in this project contains audio recordings of digits (0–9) spoken by both male and female speakers. The audio files were initially organized into separate folders for male and female voices. For the purpose of this project, these folders were merged, and the files were reorganized into folders named `d0`, `d1`, ..., `d9`, where each folder contained audio files for a specific digit. This adjustment allowed for more straightforward access to the data and simplified the file naming convention.

2.1.1 Dataset Structure

The final structure of the dataset after reorganization is as follows:

```
Dataset/
    d0/ (Class 0: Zero)
        baab_d0_01.wav
        baab_d0_02.wav
        ...
    d1/ (Class 1: One)
        baab_d1_01.wav
        baab_d1_02.wav
        ...
    ...
    d9/ (Class 9: Nine)
        baab_d9_01.wav
        baab_d9_02.wav
        ...
```

This standardized structure facilitated data loading and preprocessing, allowing for efficient implementation of the system's pipeline.

2.1.2 Filename Normalization

To ensure consistency in the dataset, several filename normalization steps were applied. The standard filename format was agreed upon as follows:

- The first four characters of the filename should consist of the first two letters from the speaker's last name, followed by the first two letters of their first name.
- The digit label should follow the pattern `_dX`, where `X` is a digit from 0 to 9.
- The sequence number should be in the format `_YY`, where `YY` is a two-digit number, ranging from 01 to 10.

However, the original filenames did not always follow this standard.Some files used `d00` through `d09`, which were inconsistent with the expected format. Additionally, there were instances of duplicated underscores `_` and the use of hyphens `-` instead of underscores.

The following Python code was used to address some of these issues:

Listing 2.1: Python code for filename normalization

```
import os
import re

# Path to the dataset
dataset_path = r"C:\Users\Dima D'origine\Downloads\RAP EXAM TP\Dataset"

# Iterate through each folder (e.g., d0, d1, ..., d9)
for folder in os.listdir(dataset_path):
    folder_path = os.path.join(dataset_path, folder)

    # Normalize folder names (e.g., d00 -> d0)
    normalized_folder = re.sub(r"d0+(\d)", r"d\1", folder)
    normalized_folder_path = os.path.join(dataset_path, normalized_folder)
    if folder != normalized_folder:
        os.rename(folder_path, normalized_folder_path)
        print(f"Renamed folder: {folder} -> {normalized_folder}")

```

```
    # Iterate through files in the folder
    for filename in os.listdir(folder_path):
        file_path = os.path.join(folder_path, filename)

        # Correct the format of dX (e.g., d09 -> d9)
        normalized_filename = re.sub(r"_d0([1-9])_",
            r"_d\1_", filename)

        # Ensure the second number after dX_ is always two
            digits (e.g., 9 -> 09)
        normalized_filename =
            re.sub(r"_d([0-9])_([0-9]{1})\.wav",
            r"_d\1_0\2.wav", normalized_filename)

        # Replace duplicate underscores with a single
            underscore and fix hyphen to underscore
        normalized_filename = re.sub(r"__+", "_",
            normalized_filename)  # Replace multiple
            underscores
        normalized_filename = re.sub(r"-", "_",
            normalized_filename)   # Replace hyphen with
            underscore

        # Rename files
        normalized_filename = normalized_filename.strip()
        normalized_filename = normalized_filename[:8] + "_d"
            + normalized_filename[10:]
        new_file_path = os.path.join(folder_path,
            normalized_filename)

        # Rename the file if it has been normalized
        if filename != normalized_filename:
            os.rename(file_path, new_file_path)
            print(f"Renamed file: {filename} ->
                {normalized_filename}")
```

These steps ensured that all filenames followed the agreed-upon standard, making it easier to load and process the data programmatically.

In addition to the changes reflected in the code above, several further normalization steps were applied to the dataset:

- Extra prefixes before the four-letter speaker identifier were removed (e.g., "ex2_baab_d0_01.wav" became "baab_d0_01.wav").

- Some files presented digits in English terms (e.g., "t1" for digit "three" and "1" for the first repetition).
- Some filenames had the four-letter identifier separated by an underscore (e.g., `ab_ad_d3_06`). Instead, it should be concatenated as `abad_d3_06`.

These additional normalization steps ensured consistency across the dataset, which was essential for the effective training and testing of the recognition model.

2.2 Data Preprocessing

To prepare the audio data for input into the model, several preprocessing steps were applied:

- **Audio Loading**: The audio files were loaded using the `librosa` library, with a sample rate of 16,000 Hz, which is a common choice for speech recognition tasks.
- **MFCC Extraction**: Mel-Frequency Cepstral Coefficients (MFCCs) were extracted from the audio data, as they are widely used features in speech recognition. MFCCs capture the essential characteristics of speech, such as pitch and timbre, which are important for digit classification.
- **Padding/Truncating**: Since audio files vary in length, they were padded or truncated to a fixed length corresponding to 1 second of audio (16,000 samples). Audio files shorter than this length were padded with zeros, while longer files were truncated.
- **Normalization**: The MFCC features were normalized using the mean and standard deviation calculated from the training set to ensure that the features had zero mean and unit variance. This step is important for ensuring consistent feature scaling and improving model performance.

2.3 Model Architecture

The core of the system is a Long Short-Term Memory (LSTM) neural network. LSTM models are particularly well-suited for sequential data, making them an ideal choice for speech recognition tasks, where the input data is sequential in nature (i.e., time-series audio features).

The model architecture is as follows:

- **Input Layer**: The input layer accepts MFCC features with a shape of `(None, 13)`, where `None` represents the variable time length of the input sequence, and 13 represents the number of MFCC features.
- **Bidirectional LSTM Layer**: A bidirectional LSTM layer with 64 units was added to capture both forward and backward dependencies in the sequential data.
- **Flatten Layer**: The output from the LSTM layer was flattened to feed into the fully connected layers.
- **Dense Layer**: A dense layer with 64 units and ReLU activation was used to introduce non-linearity and learn higher-level representations.
- **Dropout Layer**: A dropout layer with a rate of 0.5 was added to prevent overfitting by randomly setting a fraction of input units to zero during training.
- **Output Layer**: The final output layer consists of 10 units (one for each digit from 0 to 9) with softmax activation, allowing the model to output a probability distribution over the 10 possible digit classes.

2.4 Training the Model

The model was trained using the training set, with 70% of the data used for training and 30% for validation. The following parameters were used during training:

- **Loss Function**: Categorical Cross-Entropy, as this is a multi-class classification problem.
- **Optimizer**: Adam optimizer, known for its efficiency and good performance in deep learning tasks.
- **Evaluation Metric**: Accuracy, to measure the percentage of correct predictions on the validation set.
- **Epochs**: The model was trained for 30 epochs with a batch size of 32.

The training process was monitored using both the training and validation accuracy to ensure that the model was not overfitting and was generalizing well to unseen data.

2.5 GUI Implementation

A graphical user interface (GUI) was developed using the `Tkinter` library to allow users to interact with the system. The GUI includes the following features:

- **Record Button**: Users can record their voice, which will be processed by the model to predict the spoken digit.
- **Upload Button**: Users can upload an audio file for prediction instead of recording in real-time.
- **Real-time Plot**: A real-time waveform of the recorded audio is displayed as the user records their voice, providing visual feedback.
- **Prediction Display**: After the audio is processed, the predicted digit is displayed in the GUI.

This GUI makes the system user-friendly and interactive, allowing users to easily test the model with their own voice recordings or uploaded audio files.

Chapter 3

System Code and Implementation

3.1 Overview of the System Architecture

The system is composed of three main components:

- **Preprocessing Module**: This component handles the loading of audio data, feature extraction (MFCCs), and normalization of the data.
- **Model Module**: This component contains the LSTM model for classification of spoken digits and gender prediction.
- **GUI Module**: This component provides a graphical interface for the user to record their voice or upload an audio file for digit prediction.

3.2 Code Structure

The code is organized into the following main files:

- `data_preprocessing.py`: Handles the audio data loading, MFCC feature extraction, and normalization.
- `model.py`: Defines and trains the LSTM model for digit and gender classification.
- `gui.py`: Implements the graphical user interface, including buttons for recording and uploading audio.
- `main.py`: The main script that integrates all components and runs the application.

3.3 Code Implementation

3.3.1 Preprocessing (Data Loading and Feature Extraction)

The audio files are loaded, and Mel-Frequency Cepstral Coefficients (MFCCs) are extracted using the `librosa` library. The features are then normalized for consistent input to the model.

Listing 3.1: Audio Loading Preprocessing and Recording Functions

```
import numpy as np
import librosa
import wave
import pyaudio

# Function to load and preprocess audio files using MFCC
def load_and_preprocess_audio(file_path, sr=16000, dt=1.0):
    audio, _ = librosa.load(file_path, sr=sr)  # Load audio file with given sample rate
    target_length = int(sr * dt)  # Target length for the audio based on the duration (dt)

    if len(audio) < target_length:
        pad_width = target_length - len(audio)  # If the audio is shorter, pad with zeros
        audio = np.pad(audio, (0, pad_width), 'constant')
    else:
        audio = audio[:target_length]  # Otherwise, trim the audio to the target length

    mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=13, n_fft=512, win_length=400, hop_length=160)  # Extract MFCC features
    return audio, mfcc.T  # Return both audio signal and MFCC features
```

3.3.2 Model (LSTM Network Implementation)

The core of the system is an LSTM model, designed to classify the spoken digits based on the MFCC features. Below is the architecture of the LSTM model used.

Listing 3.2: LSTM Model Architecture

```
from tensorflow.keras import layers, models

# Function to create the LSTM model
def create_lstm_model(n_classes=10):
    model = models.Sequential()
    model.add(layers.Input(shape=(None, 13)))
    model.add(layers.Bidirectional(layers.LSTM(64)))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(n_classes, activation='softmax'))
    model.compile(optimizer='adam',
        loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

3.3.3 GUI (Graphical User Interface)

The GUI is implemented using the `Tkinter` library, allowing the user to interact with the system. Below is an example of how the user can record or upload an audio file for digit prediction.

Listing 3.3: GUI for Recording and Uploading Audio

```
import tkinter as tk
from tkinter import filedialog
from matplotlib.backends.backend_tkagg import
    FigureCanvasTkAgg
import matplotlib.pyplot as plt
from audio_processing import record_audio,
    load_and_preprocess_audio
import numpy as np

# Function to handle recording button click event
def on_record_button_click(model, mfcc_mean, mfcc_std,
    result_label, plot_ax, root, stop_event):
    audio_file = record_audio(duration=3, plot_ax=plot_ax,
        root=root)  # Record 3-second audio
    processed_audio, _ = load_and_preprocess_audio(audio_file)
    processed_audio = (processed_audio - mfcc_mean) / mfcc_std
    processed_audio = np.expand_dims(processed_audio, axis=0)
        # Add batch dimension
    prediction = model.predict(processed_audio)
```

```
    predicted_digit = np.argmax(prediction)
    result_label.config(text=f"Predicted digit:
       {predicted_digit}")

# Create Tkinter GUI
root = tk.Tk()
root.title("Speech Digit Recognition System")

result_label = tk.Label(root, text="Predicted digit: ",
   font=("Arial", 14))
result_label.pack()

plot_fig, plot_ax = plt.subplots()
canvas = FigureCanvasTkAgg(plot_fig, master=root)
canvas.get_tk_widget().pack()

record_button = tk.Button(root, text="Record Audio",
   command=lambda: on_record_button_click(model, mfcc_mean,
   mfcc_std, result_label, plot_ax, root, stop_event))
record_button.pack()

upload_button = tk.Button(root, text="Upload Audio")
upload_button.pack()

root.mainloop()
```

Chapter 4

GUI Development

The graphical user interface (GUI) is an essential component of the speech recognition system, allowing users to interact with the model in a convenient and intuitive way. The GUI provides features for recording speech, uploading audio files for prediction, displaying real-time waveforms, and showing prediction results.

The development of the GUI was done using Python and the Tkinter library. The interface is simple and user-friendly, featuring the following elements:

- **Record Button**: Starts and stops the recording of speech.
- **File Upload**: Allows the user to upload an audio file for prediction.
- **Prediction Display**: Shows the predicted digit.
- **Real-Time Waveform**: Displays the waveform of the recorded audio while recording.

The GUI ensures that the user can easily interact with the speech recognition system, making the testing process more accessible.

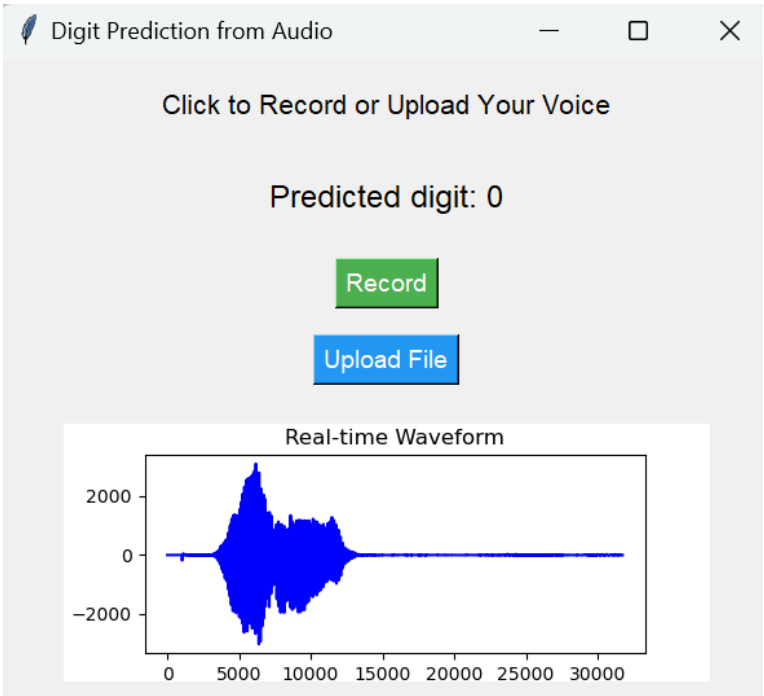

Figure 4.1: Screenshot of the GUI displaying real-time waveform during recording

The main components of the GUI were designed to handle real-time interaction, ensuring smooth recording and prediction with minimal latency.

4.1 Real-Time Waveform Display

One of the key features of the GUI is the ability to display the real-time waveform of the audio as it is being recorded. This provides immediate feedback to the user, allowing them to visualize the sound they are recording. This feature was implemented using the Tkinter Canvas and integrated with the audio recording functionality. The waveform is drawn in real-time as the audio is captured, making it easier for users to monitor the quality and duration of their recordings.

Chapter 5

System Evaluation

The evaluation of the speech recognition system is essential for measuring its performance and effectiveness. This section presents the different evaluation methods used to test the classification model, as well as the results obtained.

5.1 Evaluation Methods

To evaluate the performance of the speech recognition model, several metrics were used. The main metrics include:

- **Accuracy**: The proportion of correct predictions among all the predictions made.
- **Loss**: Measures the error of the model during training and validation. A low loss indicates the model's ability to predict accurately.
- **Accuracy and Loss Curves**: The accuracy and loss plots over epochs help visualize the model's behavior over time.

These metrics were used to test the effectiveness of the LSTM (Long Short-Term Memory) model and identify areas for improvement in the system.

5.2 Tests and Results

Tests were performed on the entire evaluation dataset, taking into account different conditions, such as variations in speech speed, recording quality, and pronunciation variations of the digits. The results obtained are summarized below.

5.2.1 Model Accuracy

The accuracy of the model is a key measure of its performance. After training the LSTM model, the accuracy obtained on the test set was 96

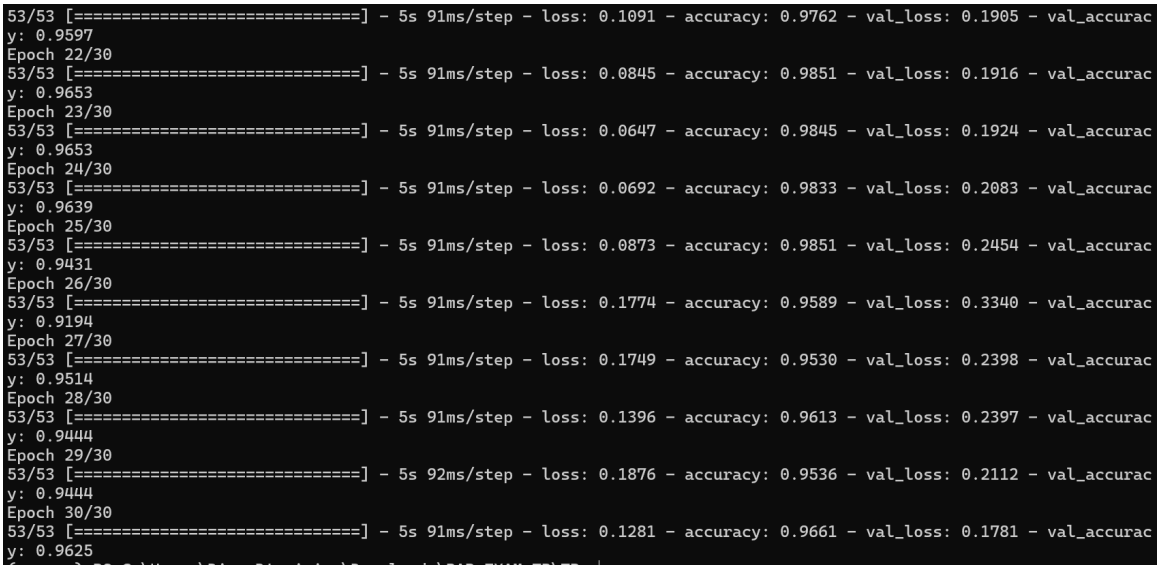

Figure 5.1: Accuracy displayed in the command line interface, showing a 96% accuracy rate

5.2.2 Model Loss

Loss was also used to track the model's improvement. The loss curve shows a continuous reduction in error over training epochs. The final loss value on the validation set was 12%. This indicates that the model has been improving over time, minimizing errors with each epoch.

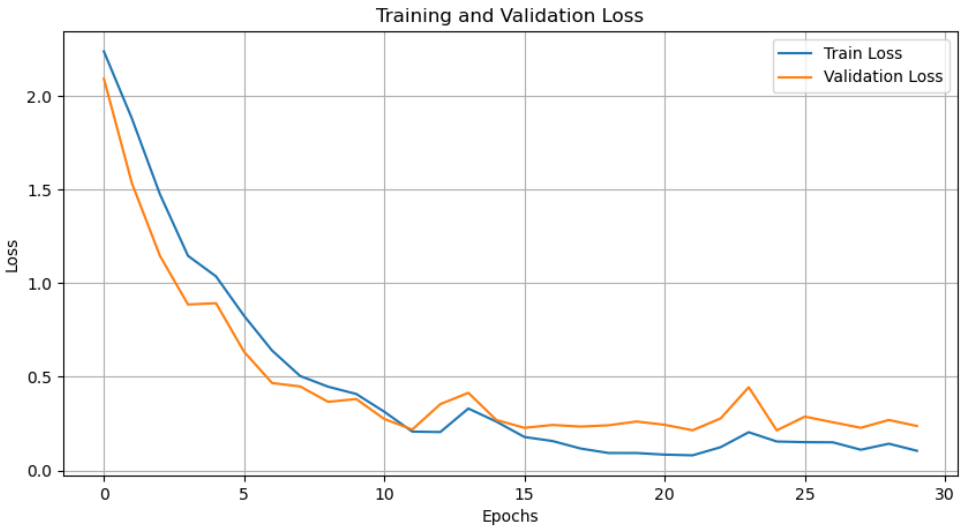

Figure 5.2: Model loss curve over epochs for LSTM

5.2.3 Model Accuracy Plot

The accuracy plot shows the progression of the model's accuracy during training across the epochs. It provides a visual representation of how the model's accuracy improved over time as it learned from the data.

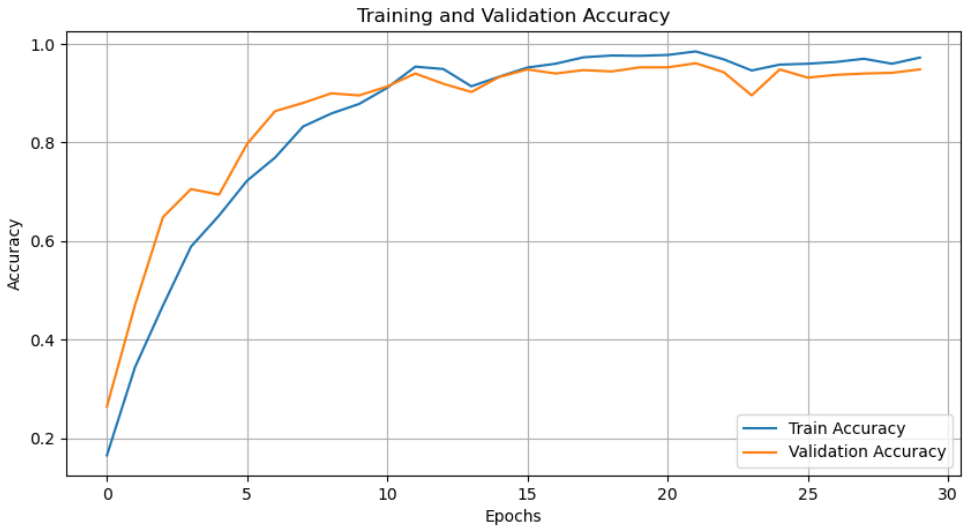

Figure 5.3: Model accuracy curve over epochs for LSTM

5.2.4 Results Analysis

The results indicate that the LSTM model performs well in recognizing digits spoken by different speakers. However, some classification errors can be attributed to external factors such as:

- **Pronunciation Variations**: Differences in how the digits are pronounced by speakers can lead to errors.
- **Recording Quality**: Poor-quality recordings or background noise can affect the model's accuracy.
- **Speech Speed**: The speed at which digits are spoken may also influence the system's performance.

5.2.5 Evaluation Conclusion

The system's evaluation shows that the LSTM model works well for digit recognition, with a satisfactory accuracy of 96% and a significant reduction in loss. Classification errors can be reduced by improving the quality of training data, increasing the dataset size, and applying signal processing techniques to reduce noise.

The results also suggest that the model could be improved by including additional data from speakers with varying accents or using advanced audio preprocessing techniques.

Chapter 6

System Demonstration

To provide a comprehensive understanding of the system's functionality, two demonstration videos are available. These videos highlight different aspects of the system's capabilities:

- **Video: predict_record**: Demonstrates the recording functionality, where the user records their voice, and the system predicts the digit in real time.
- **Video: predict_uploaded_file**: Shows how the system processes an uploaded audio file for digit prediction.

Access the demonstration videos here: System Demonstration Videos

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This project aimed to design and develop a speech recognition system capable of recognizing spoken digits from 0 to 9. Using the Long Short-Term Memory (LSTM) model, we successfully built a system that achieves an accuracy of 96% on the test set. The system demonstrated robustness in handling variations in speech speed, pronunciation, and recording quality, highlighting the effectiveness of the model and preprocessing techniques.

The GUI provides a user-friendly interface, allowing users to record audio, upload pre-recorded audio files, and obtain predictions in real-time. The system achieves the goal of recognizing digits efficiently, making it suitable for practical applications.

7.2 Future Work

While the current system performs well, there are several areas for improvement and expansion:

- **Generalization**: To make the system more robust, it could be trained on a dataset that includes greater variation in speaker profiles, such as age and accents.
- **Gender and Age Prediction**: The system could be extended to predict not only the spoken digit but also the speaker's gender and age, adding new dimensions to its capabilities.
- **Expanding Digit Range**: Currently limited to digits 0 to 9, the system could be expanded to recognize larger ranges of numbers.

- **Enhancing the GUI**: Additional features such as real-time waveform display during audio recording, more visualization options, and multilingual support could significantly improve user experience.
- **Integration with Real-World Systems**: Deploying the system in real-world environments such as voice-controlled devices or accessibility tools would provide practical insights and drive further enhancements.
- **Optimization**: Reducing the model's prediction time while maintaining accuracy would increase usability in applications requiring real-time responses.

7.3 Personal Perspective

From my viewpoint, this project showcases the potential of machine learning models in speech recognition. While achieving high accuracy, the system still has room for improvement, particularly in terms of handling diverse speaker conditions. I believe the suggestions above, such as integrating age and gender prediction, expanding the dataset, and enhancing the GUI, could take this system to the next level.

Overall, this project was a valuable learning experience, and it has laid the groundwork for further innovations in the field of speech recognition.