PRESIDENCY UNIVERSITY

Private University Estd. in Karnataka State by Act No. 41 of 2013

BANGALORE

A Project Report

On

Real Time Accent Translation

Sl. No.	Roll Number	Student Name
1	20211CSG0011	ZAINAB HANA
2	20211CSG0021	PAWAN P
3	20211CSG0022	RISHITH R RAI
4	20211CSG0027	RAKSHITHA S
5	20211CSG0031	GANASHREE P

School of Computer Science,

Presidency University, Bengaluru.

Under the guidance of,

Dr.Saravana Kumar (Assistant Professor)

School of Computer Science,

Presidency University,

Bengaluru

ABSTRACT

In an increasingly globalized world, language and regional accents can pose communication barriers, especially in real-time conversations. The "Real-Time Accent Translation" project aims to bridge this gap by developing an innovative system that translates spoken words from one accent to another while preserving the original language and meaning. Leveraging cutting-edge advancements in speech recognition, natural language processing (NLP), and machine learning, the system identifies regional accents in real-time audio input and converts them into a neutral or user-specified accent for improved comprehension.

This project incorporates deep learning models such as recurrent neural networks (RNNs) and transformers for accurate phoneme recognition and accent adaptation. The solution is designed to be platform-agnostic, deployable in mobile applications, video conferencing tools, and assistive devices, thereby enhancing accessibility and inclusivity. The system also integrates noise-cancellation techniques to ensure performance in dynamic environments. Potential applications include international business communication, education, customer service, and entertainment. By facilitating clearer communication across diverse accents, the project contributes to breaking down linguistic barriers and fostering global understanding.

INTRODUCTION

- In our increasingly globalized world, effective communication is crucial, especially during conference calls where people from different countries and backgrounds come together. As businesses grow internationally, the need for clear and smooth communication becomes even more important. However, language barriers, particularly differences in accents, can create misunderstandings and slow down conversations, impacting productivity.
- While technology has made remote collaboration easier, it hasn't fully solved the challenges posed by various accents. Participants in conference calls often find it hard to understand each other because of variations in pronunciation and speech patterns. This can lead to frustration and confusion, as traditional translation tools may not adequately address the accent differences.
- To tackle this problem, our project aims to develop a real-time Accent Translation system that enhances communication during conference calls .This innovative solution will use advanced technologies like speech recognition and machine learning to ensure clear conversations. By detecting a speaker's accent and translating their speech into the listener's preferred language and accent, the system will help overcome communication barriers in multicultural settings.
- The system will consist of several integrated components, including an accent detection module, a speech-to-text engine, a translation engine, and a text-to-speech system. These parts will work together to provide accurate, real-time translations, allowing participants to engage in meaningful discussions without the difficulties caused by language differences. Additionally, the system will offer user-friendly features, enabling users to customize their language and voice preferences for more personalized experience.
- The goals of this project go beyond just translation; they include improving overall communication effectiveness, reducing delays, and creating a collaborative environment where everyone can share their thoughts freely. By incorporating feedback from users, the system will continually improve based on real-world interactions, adapting to the way people speak.

ALGORITHM DETAILS

1.

Objective:

- The primary goal of the provided code is to implement a real-time accent recognition and speech transcription system. It processes audio streams received from a client, predicts the accent of the speaker using a trained deep learning model, and transcribes the spoken words into text. The results, including the detected accent and transcription, are sent back to the client for further use, enabling seamless and accent-adaptive communication.

Key Components:

1.Flask Server and SocketIO:

- Flask: Serves as the backend framework to create the server that processes incoming audio data.

- SocketIO: Facilitates real-time, bidirectional communication between the server and clients (e.g., a web browser).

2.Deep Learning Model:

- A pre-trained TensorFlow/Keras model (trained_model.h5) is used to predict accents.

- Features are extracted using MFCC (Mel-Frequency Cepstral Coefficients), commonly used for audio processing.

3. Label Encoder:

- Maps numerical model outputs (class indices) back to human-readable accent labels.

- The classes are preloaded from a saved file (classes.npy).

4.SpeechRecognition Library:

- Converts audio into text using Google's Speech-to-Text API for transcription.

5.Audio Processing:

- Raw audio chunks are processed into a format suitable for both model input (accent recognition) and transcription.

Process:

- The real-time accent recognition and transcription system operates by integrating several components into a seamless workflow. The server, built using Flask and enhanced with SocketIO for real-time communication, initializes by loading a pre-trained TensorFlow model and a corresponding label encoder that maps numerical predictions to human-readable accent labels. When the server receives an audio chunk from the client, the raw audio data is first converted into a NumPy array for processing.

- For accent recognition, the system extracts Mel-Frequency Cepstral Coefficients

(MFCC) features using the librosa library, which capture the essential characteristics of the audio signal. These features are then reshaped and passed to the loaded deep learning model to predict the accent, with the numerical output translated into an accent label using the label encoder. Simultaneously, the audio is transcribed into text using the SpeechRecognition library, where the audio data is converted into a WAV format and processed by Google's Speech-to-Text API.

- The server combines the results from both processes—the detected accent and the transcribed text—and emits them back to the client in real-time. Robust error handling ensures that any processing issues are logged, and appropriate error messages are sent to the client. This workflow ensures efficient and accurate recognition and transcription, making the system suitable for applications requiring seamless, accent-aware communication.

SOURCE CODE DETAILS

backend(flask server):

```
import numpy as np
import io
from flask import Flask, request
from flask_socketio import SocketIO
import speech_recognition as sr
import tensorflow as tf
from sklearn.preprocessing import LabelEncoder

app = Flask(_name_)
app.config['SECRET_KEY'] = 'fd3fa9121ad61fd26b82590fb712f0c3e64ba1a6f123b818'
socketio = SocketIO(app, cors_allowed_origins="*")

# Load the pre-trained model and label encoder
model = tf.keras.models.load_model('model/trained_model.h5')  # Update path
label_encoder_classes = np.load('model/classes.npy', allow_pickle=True)
label_encoder = LabelEncoder()
label_encoder.classes_ = label_encoder_classes

def extract_features_from_audio(audio_data, sample_rate=16000):
    """Extract MFCC features from audio data."""
    import librosa
    mfccs = librosa.feature.mfcc(y=audio_data, sr=sample_rate, n_mfcc=13)
    return np.mean(mfccs.T, axis=0)

def recognize_accent(audio_data, sample_rate):
    """Predict the accent from audio data."""
    mfcc_features = extract_features_from_audio(audio_data, sample_rate)
    mfcc_features = mfcc_features.reshape(1, -1)  # Reshape for model input
    prediction = model.predict(mfcc_features)
    predicted_class = np.argmax(prediction, axis=1)
    accent = label_encoder.inverse_transform(predicted_class)
    return accent[0]

@socketio.on('audio_chunk')
```

```
def handle_audio_chunk(audio_chunk):
    try:
        # Convert raw audio data (which should be a Float32Array from the worklet) to NumPy array
        # Assuming audio_chunk is sent as a Float32Array or similar format
        audio_data = np.frombuffer(audio_chunk, dtype=np.float32)

        # Perform accent detection
        accent = recognize_accent(audio_data, 16000)

        # Transcribe the audio using SpeechRecognition
        recognizer = sr.Recognizer()
        with io.BytesIO() as wav_buffer:
            # Convert audio data to WAV for transcription
            wav_data = np.int16(audio_data * 32767)  # Convert to PCM format (16-bit)
            wav_audio = np.array(wav_data, dtype=np.int16).tobytes()
            wav_buffer.write(wav_audio)
            wav_buffer.seek(0)
            with sr.AudioFile(wav_buffer) as source:
                audio = recognizer.record(source)
                text = recognizer.recognize_google(audio)

        # Send results back to the client
        socketio.emit('transcribed_text', {'text': text, 'accent': accent})

    except Exception as e:
        print(f"Error processing audio chunk: {e}")
        socketio.emit('transcribed_text', {'text': '', 'accent': 'Error'})

if _name_ == "_main_":
        socketio.run(app, debug=True)
```

2.

Objective:

- The primary objective of the provided React component is to create a user-friendly interface for a real-time accent recognition and transcription system. The application allows users to record audio, send it to a backend server for processing, and receive and display both the transcription of the speech and the detected accent. This facilitates real-time communication with enhanced understanding across different accents.

Key Components:

1. State Management:

- isRecording: Tracks whether audio recording is active.
- transcription: Stores the text transcription of the recorded audio.
- detectedAccent: Stores the predicted accent of the speaker.

2. **Web Audio API:**
    - Used to capture audio from the user's microphone and process it in real-time.
    - Includes components like AudioContext, MediaStreamAudioSourceNode, and AudioWorkletNode.
3. **Socket.IO:**
    - Establishes a WebSocket connection to the backend server for real-time communication.
    - Receives transcription and accent detection results via the "transcribed_text" event.
4. **Audio Processing:**
    - AudioWorkletProcessor: A custom audio processor script (/audio-processor.js) handles chunking and preprocessing audio data for transmission to the backend.
5. **User Interface:**
    - Buttons to toggle audio recording.
    - Dynamic display of transcription results and detected accent using React state.
6. **Error Handling:**
    - Manages microphone access errors and ensures resources like audio tracks and contexts are cleaned up properly.

Process:

- The process workflow begins with the initialization of the component, where a WebSocket connection is established with the backend server using Socket.IO. The application listens for "transcribed_text" events from the server, which provide the transcribed text and the detected accent in real time.
- When the user clicks the "Start Recording" button, the app initializes the Web Audio API by creating an AudioContext to manage audio streams and processing. It accesses the user's microphone using navigator.mediaDevices.getUserMedia and connects the audio input to a custom AudioWorkletNode, which processes the audio in chunks. These chunks are then transmitted to the server via the WebSocket for further processing.
- On the server side, the audio chunks are analyzed to predict the speaker's accent and transcribe the speech. The results are sent back to the client, where the "transcribed_text" event handler updates the React state variables transcription and detectedAccent. These updates are displayed dynamically in the user interface, providing real-time feedback to the user.
- When the user clicks "Stop Recording," the app stops all microphone tracks, closes the AudioContext to free up resources, and resets the recording state. Proper cleanup is also ensured during component unmount to remove WebSocket event listeners and prevent memory leaks, creating a seamless and efficient workflow for real-time accent recognition and transcription.

SOURCE CODE DETAILS

```
import React, { useState, useRef, useEffect } from "react";
import { io } from "socket.io-client";
import "./app.css";

const socket = io("http://localhost:5000"); // Replace with your Flask server's URL
```

```
function App() {
  const [isRecording, setIsRecording] = useState(false);
  const [transcription, setTranscription] = useState("");
  const [detectedAccent, setDetectedAccent] = useState("");
  const audioContextRef = useRef(null); // Web Audio API context
  const microphoneRef = useRef(null); // Microphone stream reference
  const audioWorkletNodeRef = useRef(null); // AudioWorkletNode reference
  const audioProcessorRef = useRef(null); // Custom audio processor script

  useEffect(() => {
    // Listen for responses from the server
    socket.on("transcribed_text", (data) => {
      setTranscription(data.text || "Could not transcribe");
      setDetectedAccent(data.accent || "Unknown");
    });

    return () => {
      socket.off("transcribed_text"); // Clean up listeners on unmount
    };
  }, []);

  const startRecording = async () => {
    try {
      // Set up Web Audio API
      const audioContext = new (window.AudioContext ||
        window.webkitAudioContext)();
      audioContextRef.current = audioContext;

      // Load the custom audio processor for the AudioWorkletNode
      await audioContext.audioWorklet.addModule("/audio-processor.js"); // Path to your
custom AudioWorkletProcessor

      // Create a MediaStreamAudioSourceNode
      const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
      microphoneRef.current = stream;

      const source = audioContext.createMediaStreamSource(stream);

      // Create the AudioWorkletNode
      audioWorkletNodeRef.current = new AudioWorkletNode(
        audioContext,
        "audio-processor"
      );

      // Connect the source to the audio worklet node
      source.connect(audioWorkletNodeRef.current);
      audioWorkletNodeRef.current.connect(audioContext.destination); // Connect to the
```

```
context's output (speakers)

    // Set up custom processor to send audio data to the server
    audioWorkletNodeRef.current.port.onmessage = (event) => {
      if (event.data) {
        sendAudioData(event.data); // Send audio data to the server
      }
    };

    setIsRecording(true);
  } catch (error) {
    console.error("Error accessing microphone:", error);
  }
 };

 const stopRecording = () => {
  if (microphoneRef.current) {
    microphoneRef.current.getTracks().forEach((track) => track.stop()); // Stop all tracks
  }

  if (audioContextRef.current) {
    audioContextRef.current.close(); // Close the AudioContext
  }

  setIsRecording(false);
 };

 const sendAudioData = (audioData) => {
  // Send audio chunk to server
  // You can convert audioData to a suitable format (e.g., WebM/Opus) before sending
  socket.emit("audio_chunk", audioData);
 };

 return (
  <div>
    <div className="nav">
      <h1 className="heading">RAT: Real-time Accent Translation</h1>
    </div>
    <div className="sidebar-left">
      <h2 className="click">Click to connect</h2>
      <button
        className="button"
        onClick={isRecording ? stopRecording : startRecording}
      >
        {isRecording ? "Stop Recording" : "Start Recording"}
      </button>
    </div>
    <div className="sidebar">
```

```
      <div className="sideblock">
        <h4>You are saying</h4>
        <p>{transcription}</p>
      </div>
      <div className="sideblock">
        <h4>Accent detected</h4>
        <p>{detectedAccent}</p>
      </div>
    </div>
  </div>
 );
}
export default App;
```

3.

Objective:

- The objective of the given code is to build and train a machine learning model capable of recognizing accents based on audio features. The system leverages **Mel-Frequency Cepstral Coefficients (MFCC)** to extract meaningful audio features, encodes labels for classification, and trains a neural network to classify the audio data into different accents.

Key Components:

1. **Feature Extraction:**
   - MFCC: Extracts 13-dimensional MFCC features from audio files to represent the key characteristics of the audio signal. These features are robust to variations in speech and are widely used in speech processing tasks.
2. **Data Loading:**
   - Audio files and labels are loaded from a dataset, using metadata provided in a validated.tsv file. Each row in this file associates an audio file with its corresponding accent label.
3. **Preprocessing:**
   - Label Encoding: Converts string labels (accents) into numerical format using LabelEncoder for compatibility with the neural network.
   - Data Splitting: Splits the dataset into training and testing sets for model evaluation.
4. **Model Architecture:**
   - Input Layer: Accepts MFCC features as input.
   - Dense Layers: Fully connected layers with ReLU activation capture complex patterns in the data.
   - Dropout Layer: Reduces overfitting by randomly disabling neurons during training.
   - Output Layer: Uses a softmax activation function to predict probabilities for each accent class.
5. **Model Training:**

- Loss Function: sparse_categorical_crossentropy calculates the error between predicted and actual labels.
- Optimizer: Adam optimizer with a learning rate of 0.001 ensures efficient weight updates.
- Metrics: Accuracy is used to evaluate model performance during training and testing.

6. **Model Saving:**
   - Trained model and label encoder classes are saved to disk for deployment, allowing reuse without retraining.
7. **Evaluation:**
   - The model's accuracy is tested on the holdout test set to measure its generalization ability.

Process:

- The process begins with the feature extraction phase, where each audio file is processed using the librosa library to compute Mel-Frequency Cepstral Coefficients (MFCC), a widely used feature in speech processing. The MFCC features are averaged over time to create a fixed-size vector that represents the audio. Next, the dataset is prepared by reading audio metadata from the validated.tsv file, which links each audio file to its corresponding accent label. The labels are then encoded into numerical values using a LabelEncoder to make them compatible with the neural network.
- The data is split into training and testing sets, with 80% allocated for training and 20% for testing. A neural network model is defined using TensorFlow/Keras. This model consists of fully connected layers with ReLU activation for feature learning, a dropout layer for regularization to reduce overfitting, and a softmax output layer for multi-class classification of accents.
- During the training phase, the model is trained on the processed training data for 20 epochs with a batch size of 32. The training process minimizes the categorical cross-entropy loss function using the Adam optimizer while tracking accuracy as a performance metric. Validation data is used to monitor the model's performance on unseen data during training. After training, the model and the label encoder's class mappings are saved to disk for deployment.
- Finally, the trained model is evaluated on the test data to assess its generalization ability. The test accuracy is calculated and printed as a measure of performance. This workflow enables the system to classify accents effectively and prepares the model for integration into real-world applications.

SOURCE CODE DETAILS:

```
import os
import numpy as np
import pandas as pd
import librosa
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
```

```
# Function to extract MFCC features from audio files
def extract_features(file_path):
    audio, sample_rate = librosa.load(file_path, sr=None)
    mfccs = librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=13)
    return np.mean(mfccs.T, axis=0)

# Load dataset
def load_data(data_dir):
    features = []
    labels = []

    # Load validated data
    validated_data = pd.read_csv(os.path.join(data_dir, 'validated.tsv'), sep='\t')

    for index, row in validated_data.iterrows():
        audio_file = os.path.join(data_dir, 'clips', row['path'])
        accent = row['accents']  # Use the 'accents' column for labels

        if os.path.isfile(audio_file):
            mfccs = extract_features(audio_file)
            features.append(mfccs)
            labels.append(accent)

    return np.array(features), np.array(labels)

# Load data
data_dir = 'data_dir'
X, y = load_data(data_dir)

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

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2,
random_state=42)

# Define a simple neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(256, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dropout(0.5),  # Add dropout for regularization
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(len(np.unique(y_encoded)), activation='softmax')
])


model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
```

```
        loss='sparse_categorical_crossentropy',
        metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=20, batch_size=32, validation_data=(X_test, y_test))

# Save the trained model
model.save('path/to/your/trained_model.h5')  # Update this path

# Save the label encoder classes
np.save('path/to/your/classes.npy', label_encoder.classes_)  # Update this path

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy:.2f}')
```

CONCLUSION

- In conclusion, the real-time accent translation system developed using the above codes integrates advanced techniques in speech processing, machine learning, and real-time communication. The system leverages MFCC feature extraction to capture the essential characteristics of speech and uses a neural network model to classify accents based on these features. The integration of Flask backend with WebSocket communication ensures that audio data is processed and transmitted efficiently for real-time feedback.
- The React frontend facilitates user interaction by allowing for seamless audio recording and instant display of both the transcribed speech and detected accent, enhancing user experience. The model, trained on a diverse dataset, is capable of recognizing various accents with a high degree of accuracy, making it a valuable tool for real-time accent identification and translation. The model evaluation ensures its effectiveness, while the use of dropout regularization helps prevent overfitting, improving its robustness.
- This system holds promise for applications such as speech transcription services, language learning tools, and multilingual communication platforms, where understanding accents plays a crucial role in improving accuracy and user experience. By combining speech recognition with accent detection, the system paves the way for more natural and efficient communication in multilingual and multicultural contexts.