Automatic Arabic Speech Recognition

Welcome to the documentation for Our Automatic Arabic Speech Recognition project.

Team Name : MAY-X

Team Supervisor : Dr / Samar Elbedwehy

Team Leader : Mohamed Atwan

Team Member : Youssef Khalf

Team Member : Ehab Ghallab

Team Member : Randa Hamada

Tabel Of Content

i.THE INTRODUCTION

This project aims to develop a robust system for transcribing Arabic speech into text automatically. The system is designed to handle Arabic language nuances and variations, making it suitable for diverse applications, including voice-activated systems and transcription services.

Participation in MTC-AIC2

We are proud to participate in the MTC-AIC2 and believe this system represents a valuable contribution to the field of artificial intelligence. We hope this system will contribute to improving human-machine communication and open new avenues for interaction in various domains.

Conclusion

We are confident that this system will play a significant role in promoting the use of Arabic language across various applications. We encourage you to try the system and share your feedback with us.

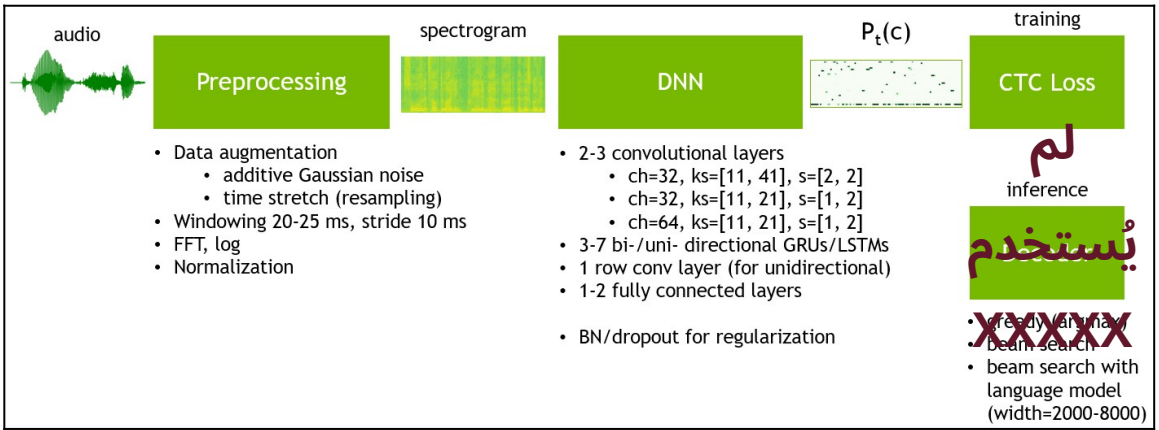
i.1 Project Overview

This project focuses on developing a speech recognition system using a “DeepSpeech 2 “ .

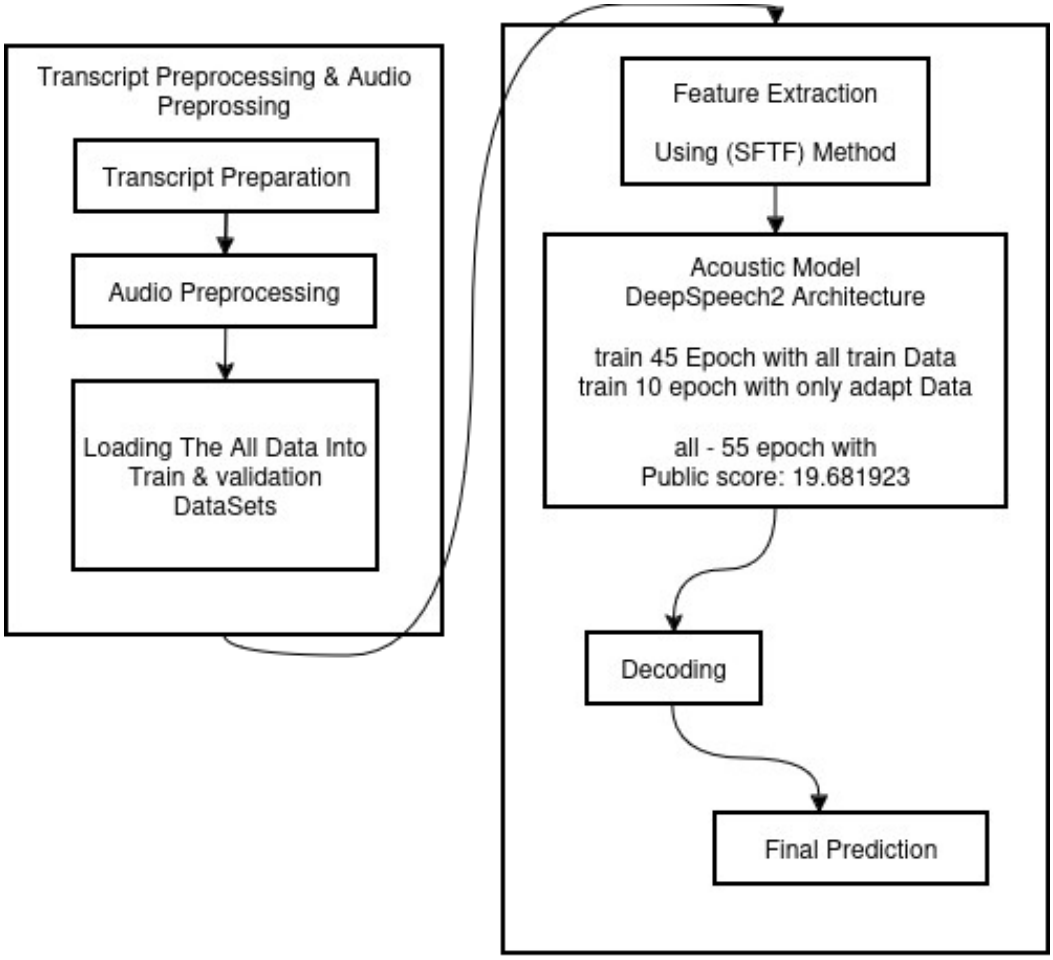
The project is organized into three main components, each crucial to achieving accurate and efficient speech recognition

i.2 Speech Recognition and DeepSpeech

- Speech recognition aims to automatically convert spoken language into text. DeepSpeech 2 achieves this by processing audio signals in the following steps:
    - **Feature Extraction:** Converts raw audio into spectrograms, which are visual representations of the frequency content over time.
    - **Model Training:** Trains a DeepSpeech 2 model using spectrograms paired with corresponding text labels. The model learns to predict the text based on the spectrogram features.
    - **Model Evaluation:** Evaluates the trained model's performance on unseen data to assess its accuracy.



i.i Workflow Diagram



i.i.i File Descriptions

i.i.i.1 TranscriptCleaning&EDA.ipynb

In this notebook, the emphasis is on cleaning and exploring the transcript data. It covers tasks such as:

- Text normalization
- Handling special characters, Nulls, Duplicated …
- Performing (EDA) to understand the characteristics of the transcript data

i.i.i.2 Audio_Preprocessing.ipynb

This notebook focuses on preparing the audio data for feature extraction and modeling. It includes steps for:

- Loading audio files
- Audio Preprocessing With Multiple Methods (Noise Reduction, Silemce Removal)
- Transforming audio signals into a format suitable for Featuers Extraction

i.i.i.3 features_extraction.py

- **Function:** Defines functions for feature extraction and character/number conversion.
- **Input:** Audio file path (string)
- **Output:**
  - Spectrogram (tensor) representing the frequency content of the audio
  - Encoded label (tensor) representing the text transcription

i.i.i.4 metrics.py

- **Function:** Defines custom metrics for the model (CTC Loss, WER, Levenshtein Distance).

- **Input:**
    - True labels (tensor) representing the ground truth text
    - Predicted labels (tensor) representing the
model's output (softmax probabilities)

- **Output:**
    - Loss value (tensor) used for model optimization during training
    - Word Error Rate (WER) score (float) indicating the percentage of errors in predicted words
    - Levenshtein distance (integer) measuring the number of edits (substitutions, insertions, deletions) needed to transform one sequence into another

i.i.i.5 model.py

- **Function:** Defines Our Model with an architecture similar to that of the “DeepSpeech 2” model architecture, a CNN followed by RNNs for speech recognition.
- **Input:** Spectrogram (tensor)
- **Output:** Predicted labels (tensor) representing the model's output (softmax probabilities) indicating the likelihood of each character in the sequence

i.i.i.6 train.py

- **Function:** Loads data, trains the model, and evaluates its performance.
- **Input:**
  - Train CSV path (string) containing audio file paths and corresponding text labels
  - Train audio directory (string) where the training audio files are located
- **Output:** Trained model weights (files) saved for future use

i.i.i.7 utils.py

- **Function:** Provides utility functions for data loading and processing.
- **Input:**
  - Train/Adaptation CSV path (string)
  - Train/Adaptation audio directory (string)
- **Output:**
  - Train/Adaptation DataFrame (pandas) containing audio paths and labels
  - Train/Validation datasets (tf.data.Dataset) ready for model training and evaluation

i.i.i.8 main.ipynb

The main notebook integrates the processed audio features with the cleaned transcripts to build and evaluate Arabic speech recognition models. It includes:

- Load The Data
- Feature extraction
- Load The training Model
- Predict With Model Any Input Audios

This documentation will guide you through each step of the process, from data preprocessing to model evaluation, helping you understand and replicate the project's outcomes effectively.

1

TranscriptCleaning&EDA

Overview

- This section focuses on preprocessing Arabic transcripts.
- The preprocessing steps include loading the data, cleaning it by removing or replacing unwanted characters, and transforming text data into numerical sequences for further processing.
- Below is a detailed explanation of each function used in the transcript preprocessing, accompanied by the code and sample outputs.

Load and Explore Transcripts

- **Purpose**: This block imports necessary libraries and loads the transcript data from a CSV file, ensuring the proper encoding for Arabic text.
- **Why Use**: Loading the data is the first step in any data preprocessing pipeline. Proper encoding ensures that the Arabic text is correctly interpreted.

- **Code:**

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the CSV file with proper encoding for Arabic text
csv_file = '/home/mohamed/Desktop/Projects/MY/Data/a/DATA/train.csv'
transcripts_df = pd.read_csv(csv_file, encoding='utf-8')
```

Displaying Initial Data

- **Purpose**: Displays the first two rows of the dataset.

- **Why Use**: Provides an initial view of the data structure, allowing you to understand the dataset's format and contents.

- **Code:**

```
transcripts_df.head(2)
```

	audio	transcript
0	train_sample_0	على إنها عار في الوقت اللي كانت بتتعامل مع أخو...
1	train_sample_1	فأكيد ربنا عوضهم خير هو الراجل بيبقى ليه إختيا...

Basic Statistics

- **Purpose**: Generates descriptive statistics of the dataset.
- **Why Use**: Helps in understanding the dataset's characteristics, such as the count, unique values, and frequency of top elements.

- **Code:**

```
transcripts_df.describe().T
```

	count	unique	top	freq
audio	50715	50715	train_sample_0	1
transcript	50709	50653	خلاص	4

Checking for Missing Values

- **Purpose**: Checks for any missing values in the dataset.
- **Why Use**: Identifies incomplete data that may need to be handled before further processing.

- **Code:**

```
transcripts_df.isnull().sum()
```

```
audio         0
transcript    6
dtype: int64
```

Dropping Missing Values

- **Purpose**: Removes rows with missing values in the 'transcript' column.
- **Why Use**: Ensures that only complete data is used for analysis, improving the accuracy of subsequent steps.

- **Code:**

```
transcripts_df.dropna(subset=['transcript'], inplace=True)
```

Character Distribution

- **Purpose**: Defines a function to count the frequency of each character in the transcripts.
- **Why Use**: Understanding character distribution is crucial for tasks such as text normalization and creating character-level models

- **Code:**

```
from collections import Counter

def character_distribution(transcripts):
    all_text = ''.join(transcripts)
    char_counter = Counter(all_text)
    return char_counter
```

Calculating Character Distribution

- **Purpose**: Calculates the character distribution and converts the results into a DataFrame for easier analysis.
- **Why Use**: Provides a detailed view of character frequencies, highlighting the diversity and commonality of characters in the transcripts.

- **Code:**

```
# Calculate character distribution
char_counter = character_distribution(transcripts_df['transcript'].tolist())

# Convert the counter to a DataFrame for better visualization
char_df = pd.DataFrame(char_counter.items(), columns=['Character', 'Frequency']).sort_values(by='Frequency', ascending=Fa

unique_classes = char_df['Character'].unique()
print(f"Unique Classes (Characters): {unique_classes}")
```

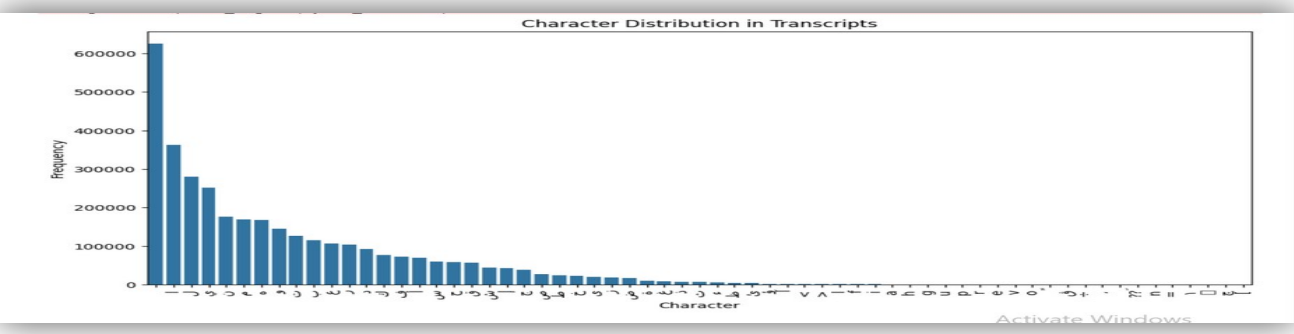
Plotting Character Distribution

- **Purpose**: Plots the character distribution using a bar chart.
- **Why Use**: Visual representations make it easier to understand and communicate the distribution of characters.

- **Code:**

```
# Plot the character distribution
plt.figure(figsize=(12, 6))
sns.barplot(x='Character', y='Frequency', data=char_df)
plt.title('Character Distribution in Transcripts')
plt.xticks(rotation=90)
plt.show()
```

Output



Most Frequent Words

- **Purpose**: Identifies and displays the 20 most frequent words in the transcripts.
- **Why Use**: Understanding common words can help in optimizing text processing steps and improving model performance.

- **Code:**

```
# Most Frequent Words
all_words = ' '.join(transcripts_df['transcript']).split()
word_counter = Counter(all_words)
most_common_words = word_counter.most_common(20)  # Top 20 most frequent words

# Convert the counter to a DataFrame for better visualization
most_common_words_df = pd.DataFrame(most_common_words, columns=['Word', 'Frequency'])

print("\nMost Frequent Words:")
print(most_common_words_df)
```

Plotting Most Frequent Words

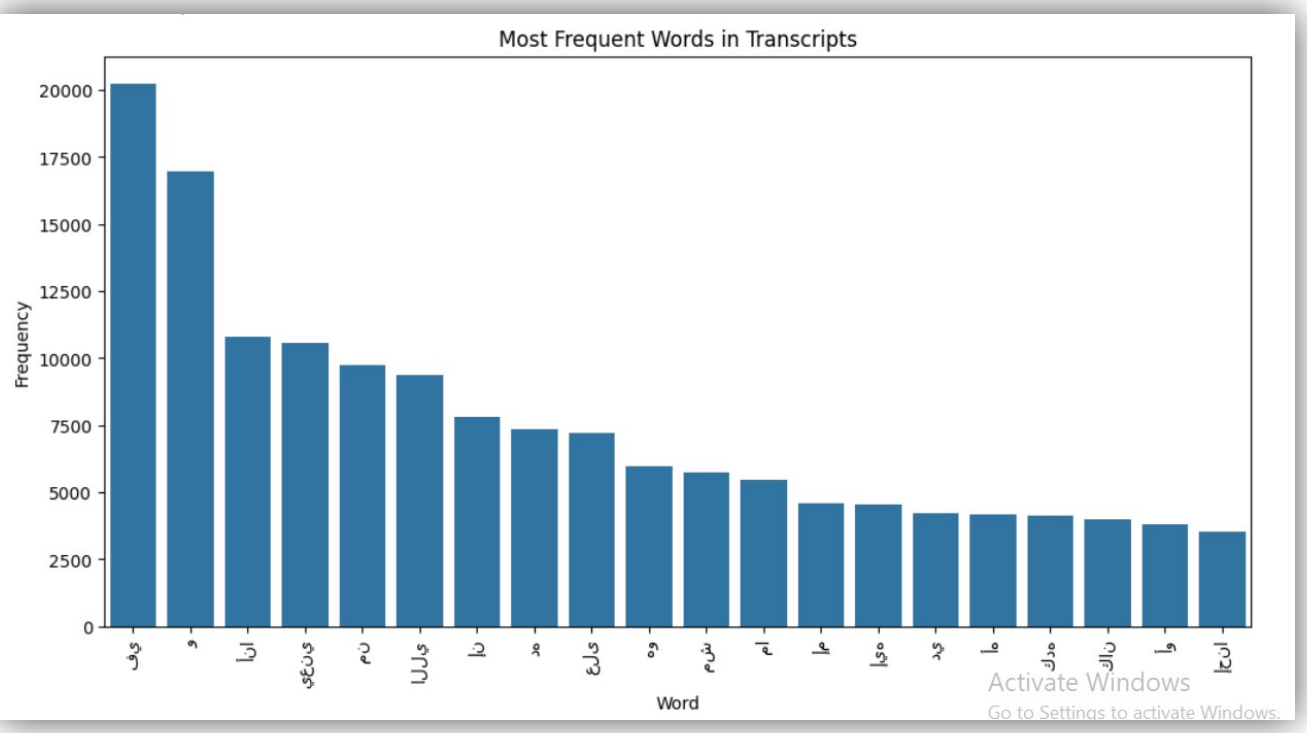
- **Purpose**: Plots the frequencies of the most common words.
- **Why Use**: Helps visualize word usage patterns, providing insights into the text's vocabulary.

- **Code:**

```
# Plot most frequent words
plt.figure(figsize=(12, 6))
sns.barplot(x='Word', y='Frequency', data=most_common_words_df)
plt.title('Most Frequent Words in Transcripts')
plt.xticks(rotation=90)
plt.show()
```

Output

>>



Word Count Distribution

- **Purpose**: Defines a function to calculate the number of words in each transcript.
- **Why Use**: Analyzing word counts helps understand the distribution of transcript lengths, which is important for model training and evaluation.

- **Code:**

```python
def word_count_distribution(transcripts):
    word_counts = [len(transcript.split()) for transcript in transcripts]
    return word_counts
```

Calculating and Plotting Word Count Distribution

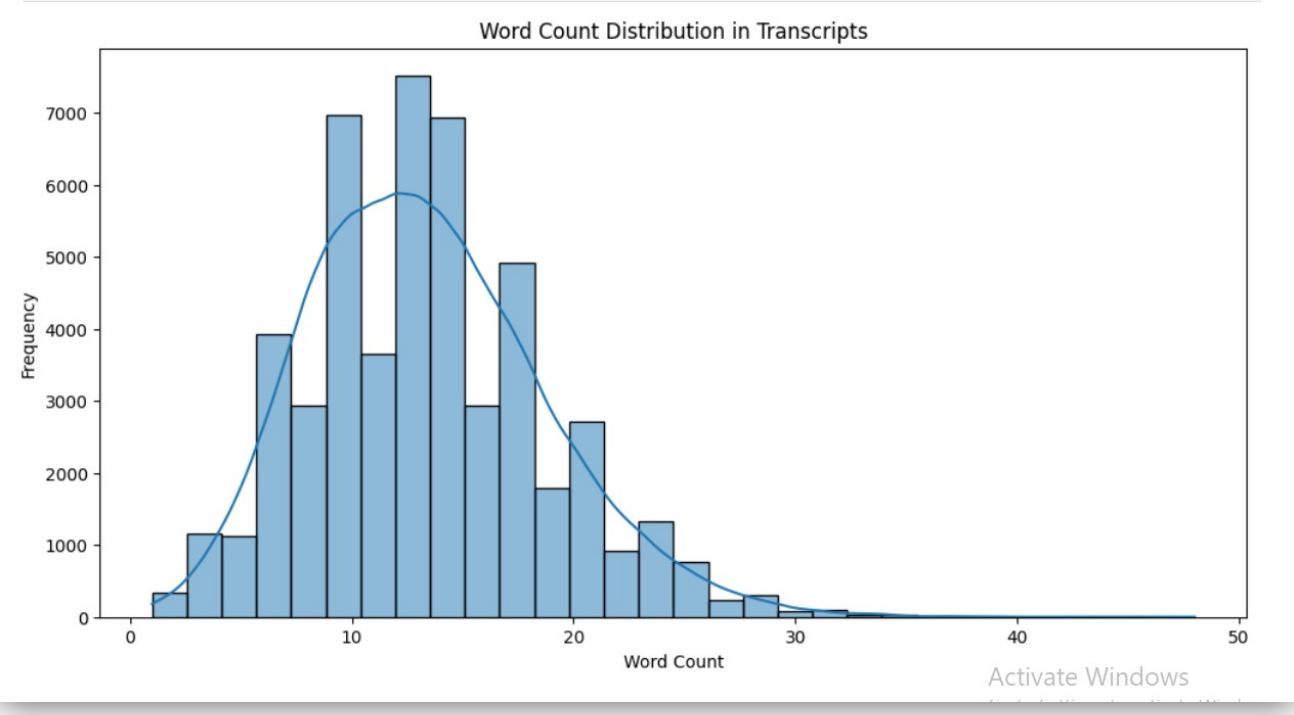
- **Purpose**: Calculates and plots the distribution of word counts in the transcripts.

- **Why Use**: Visualizing word count distribution helps identify patterns and anomalies in transcript lengths.

- **Code:**

```
def word_count_distribution(transcripts):
    word_counts = [len(transcript.split()) for transcript in transcripts]
    return word_counts

# Calculate word count distribution
word_counts = word_count_distribution(transcripts_df['transcript'].tolist())

# Plot the word count distribution
plt.figure(figsize=(12, 6))
sns.histplot(word_counts, bins=30, kde=True)
plt.title('Word Count Distribution in Transcripts')
plt.xlabel('Word Count')
plt.ylabel('Frequency')
plt.show()
```

Output



Transcript Length Analysis

- **Purpose**: Defines a function to calculate the length of each transcript.
- **Why Use**: Analyzing transcript lengths provides insights into the text's structure and complexity.
- **Code:**

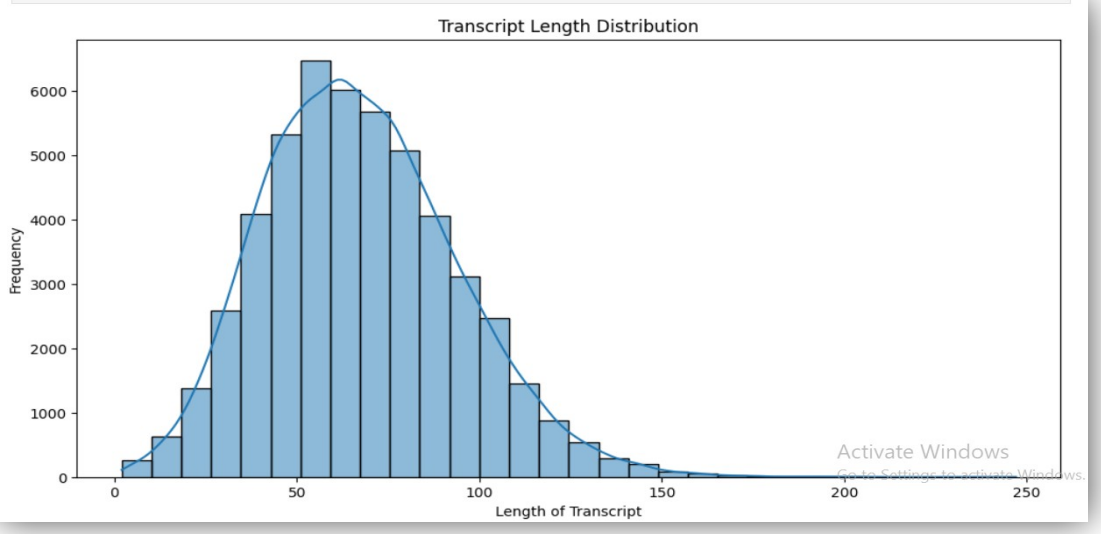
```
def transcript_length_distribution(transcripts):
    lengths = [len(transcript) for transcript in transcripts]
    return lengths
```

Calculating and Plotting Length Distribution

- **Purpose**: Calculates and plots the distribution of transcript lengths.
- **Why Use**: Helps in understanding the variability in transcript lengths, which can impact model performance.
- **Code:**

```
# Calculate transcript length distribution
transcript_lengths = transcript_length_distribution(transcripts_df['transcript'].tolist())

# Plot the transcript length distribution
plt.figure(figsize=(12, 6))
sns.histplot(transcript_lengths, bins=30, kde=True)
plt.title('Transcript Length Distribution')
plt.xlabel('Length of Transcript')
plt.ylabel('Frequency')
plt.show()
```

Output



Transcripts Cleaning

- The following code performs necessary preprocessing steps on Arabic transcript data. These steps include loading the data, handling missing values, removing duplicates, and converting text to integer sequences.

Text Transform Class

- __init__ Method
- text_to_int Method
- int_to_text Method

__init__ Method

- **Purpose:**Initializes the TextTransform class by setting up character mappings between Arabic characters and integers.
- **Why Use:**Provides a standardized way to convert Arabic text into numeric sequences, which is essential for text preprocessing in machine learning.
- **Code:**

```
class TextTransform:
    """Maps characters to integers and vice versa."""
    def __init__(self):
        char_map_str = """
        ' 0
        <SPACE> 1
        2 ا
        3 ب
        4 ت
        5 ث
        6 ج
        7 ح
        8 خ
        9 د
        10 ذ
        11 ر
        12 ز
        13 س
        14 ش
        15 ص
        16 ض
        17 ط
        18 ظ
        19 ع
        20 غ
        21 ف
        22 ق
        23 ك
        24 ل
        25 م
        26 ن
        27 ه
        28 و
        29 ي
        30 ء
        31 آ
        32 أ
        33 ؤ
        34 إ
        35 ئ
        36 ى
        """
        self.char_map = {}
        self.index_map = {}
        for line in char_map_str.strip().split('\n'):
            ch, index = line.split()
            self.char_map[ch] = int(index)
            self.index_map[int(index)] = ch
        self.index_map[1] = ' '  # Map <SPACE> to space character
```

text_to_int Method

- **Purpose:**Converts a given Arabic text string into an integer sequence based on the character mappings.
- **Why Use:**Facilitates the transformation of textual data into a numerical format suitable for machine learning models.
- **Code:**

```python
def text_to_int(self, text):
    """Converts text to an integer sequence using a character map."""
    int_sequence = []
    for c in text:
        if c == ' ':
            int_sequence.append(self.char_map['<SPACE>'])
        elif c in self.char_map:
            int_sequence.append(self.char_map[c])
        else:
            # Handle unknown characters (optional)
            int_sequence.append(0)  # Or another token for unknown characters
    return int_sequence
```

int_to_text Method

- **Purpose:**Converts a sequence of integer labels back into the corresponding Arabic text using the character mappings.
- **Why Use:**Enables the reconstruction of text from numeric sequences, useful for validating the transformation process.
- **Code:**

```python
def int_to_text(self, labels):
    """Converts integer labels to a text sequence using a character map."""
    return ''.join([self.index_map[i] for i in labels]).replace('<SPACE>', ' ')
```

preprocess_arabic_transcripts Function

- **Purpose:**Performs preprocessing on Arabic transcript data, including loading data, handling missing values, removing duplicates, and converting text to integer sequences.
- **Why Use:**Ensures that the transcript data is clean and standardized, making it suitable for further analysis or model training.

find_non_arabic Function

- **Purpose:**Finds non-Arabic characters within a given text.
- **Why Use:**Identifies characters that may need to be removed or replaced to ensure data consistency.
- **Code:**

```
def find_non_arabic(text):
    non_arabic_pattern = re.compile(r'[^\u0600-\u06FF\s]')
    non_arabic_characters = ''.join(set(re.findall(non_arabic_pattern, text)))
    return non_arabic_characters
```

replace_characters_of_interest Function

- **Purpose:**Replaces specific characters of interest with standardized Arabic characters.
- **Why Use:**Ensures consistent representation of characters in the transcript data.
- **Code:**

```
def replace_characters_of_interest(text):
    replacements = {
        'ا' :'أ',
        'ج' :'ج',
        'ف' :'ق',
        'ه': 'ة'
    }
    for old_char, new_char in replacements.items():
        text = text.replace(old_char, new_char)
    return text
```

Loading Data

- **Purpose:**Reads the CSV file containing Arabic transcripts using UTF-8 encoding.
- **Why Use:**Ensures proper loading and handling of Arabic text data.

- **Code:**

```
# Load the CSV file with proper encoding for Arabic text
transcripts_df = pd.read_csv(csv_file, encoding='utf-8')
```

Data Cleaning

- **Purpose:**Identifies and removes rows with NaN values and duplicate transcripts, and re-indexes the DataFrame.
- **Why Use:**Ensures data integrity by removing incomplete and redundant data.

- **Code:**

```
# Check for NaN values in transcript column and remove corresponding rows
nan_rows = transcripts_df[pd.isna(transcripts_df['transcript'])].index.tolist()
removed_rows = []  # To store indices of removed rows

for idx in nan_rows:
    if idx < len(transcripts_df):
        removed_rows.append(idx)
        print(f"Removed row '{transcripts_df.iloc[idx]['audio']}' due to NaN transcript")

# Drop rows with NaN transcripts
transcripts_df.dropna(subset=['transcript'], inplace=True)

# Drop duplicate transcripts
transcripts_df.drop_duplicates(subset=['transcript'], inplace=True)

# Re-index the DataFrame
transcripts_df.reset_index(drop=True, inplace=True)
```

Processing Each Transcript

- **Purpose:**Cleans the text in the transcript column by replacing characters of interest, removing unwanted characters, and converting the text to numerical sequences.
- **Why Use:**Prepares the data for modeling or analysis, ensuring that the texts follow the same format and standards.
- **Code:**

```
for idx, row in transcripts_df.iterrows():
    transcript = row['transcript']

    # Replace characters of interest
    transcript = replace_characters_of_interest(transcript)

    # Find non-Arabic characters (if needed)
    # non_arabic_characters = find_non_arabic(transcript)

    # Remove any unwanted characters not in the TextTransform
    cleaned_transcript = ''.join([c for c in transcript if c in text_transform.char_map or c == ' '])

    # Example of converting transcript to integer sequence
    int_sequence = text_transform.text_to_int(cleaned_transcript)

    # Replace original transcript with cleaned version
    transcripts_df.at[idx, 'transcript'] = cleaned_transcript
```

Saving Output

- **Purpose:**Saves the cleaned and processed DataFrame to a specified CSV file.
- **Why Use:**Ensures that the processed data is stored correctly for further use and provides feedback on removed rows.
- **Code:**

```
# Save the final edited DataFrame to a new CSV file
transcripts_df.to_csv(output_file, index=False, encoding='utf-8')

print(f"Indices of rows with NaN transcripts: {removed_rows}")
```

Displaying Transcripts

- **Purpose:**Tests the encoding and decoding process applied to the text within the transcripts_df DataFrame.
- **Why Use:**Verifies the accuracy and effectiveness of the transformations performed by the text_to_int and int_to_text functions.
- **Code:**

```python
# Print a sample of 5 encoded transcripts and decode them back
sample_transcripts = transcripts_df['transcript'].head(5)
for idx, transcript in enumerate(sample_transcripts):
    int_sequence = text_transform.text_to_int(transcript)
    decoded_text = text_transform.int_to_text(int_sequence)
    print(f"\nSample {idx + 1}:")
    print(f"Original Transcript: {transcript}")
    print(f"Encoded: {int_sequence}")
    print(f"Decoded: {decoded_text}")
```

Main Script Execution (Testing Preprocessing Function On Transcript)

- **Purpose:**Specifies file paths and initiates the preprocessing function.
- **Why Use:**Defines the workflow for running the script directly and processing the input data.
- **Code:**

```python
if __name__ == "__main__":
    csv_file = r"/home/mohamed/Desktop/Projects/MY/Data/a/DATA/adapt.csv"
    output_file = r"./Data/Clean_adapt.csv"
    df=pd.read_csv(csv_file)
    cleaned_df = preprocess_arabic_transcripts(csv_file, output_file)
```

Output

```
Indices of rows with NaN transcripts: []

Sample 1:
Original Transcript: شوفلنا المشوار ده يا حج
Encoded: [14, 28, 21, 24, 26, 2, 1, 2, 24, 25, 14, 28, 2, 11, 1, 9, 27, 1, 29, 2, 1, 7, 6]
Decoded: شوفلنا المشوار ده يا حج

Sample 2:
Original Transcript: لأ للأسف دكتوره واحده بس بتعمل العمليه ديت عندنا في المحافظه
Encoded: [24, 32, 1, 24, 24, 32, 13, 21, 1, 9, 23, 4, 28, 11, 27, 1, 28, 2, 7, 9, 27, 1, 3, 13, 1, 3, 4, 19, 25, 24, 1, 2,
24, 19, 25, 24, 29, 27, 1, 9, 29, 4, 1, 19, 26, 9, 26, 2, 1, 21, 29, 1, 2, 24, 25, 7, 2, 21, 18, 27]
Decoded: لأ للأسف دكتوره واحده بس بتعمل العمليه ديت عندنا في المحافظه

Sample 3:
Original Transcript: والراجل تبصله يعني إبن زمنه
Encoded: [28, 2, 24, 11, 2, 6, 24, 1, 4, 3, 15, 24, 27, 1, 29, 19, 26, 29, 1, 34, 3, 26, 1, 12, 25, 26, 27]
Decoded: والراجل تبصله يعني إبن زمنه

Sample 4:
Original Transcript: و أنت كيف عرفته أبترل يا عمي
Encoded: [28, 1, 32, 26, 4, 1, 23, 29, 21, 1, 19, 11, 21, 4, 27, 1, 32, 3, 4, 11, 24, 1, 29, 2, 1, 19, 25, 29]
Decoded: و أنت كيف عرفته أبترل يا عمي

Sample 5:
Original Transcript: ميجرفوتش حاجه عن السوبر أه غير إنه لب
Encoded: [25, 29, 19, 11, 21, 28, 14, 1, 7, 2, 6, 27, 1, 19, 26, 1, 2, 24, 13, 28, 3, 11, 1, 32, 27, 1, 20, 29, 11, 1, 34,
26, 27, 1, 24, 3]
Decoded: ميجرفوتش حاجه عن السوبر أه غير إنه لب
```

2
Audio_Preprocessing.ipynb

Overview

- This notebook serves as a crucial component for preparing audio data before feeding it into speech recognition models. It covers various preprocessing techniques such as noise reduction, silence removal, and data augmentation to enhance the quality and accuracy of Arabic speech recognition systems.

AudioProcessor Class

The AudioProcessor class encapsulates essential functions for audio preprocessing. Each function plays a key role in transforming raw audio data into a format suitable for machine learning models.

Function: read_audio(filepath, sr=None)

- **Purpose**: Loads an audio file from the specified filepath using librosa.
- **Why Use**: Essential for reading audio files into the notebook for further processing.
- **Code:**

```python
def read_audio(self, filepath, sr=None):
    """
    Read audio file from filepath using librosa.

    Parameters:
    - filepath (str): Path to the audio file.
    - sr (int or None): Sampling rate to load the audio file (optional).

    Returns:
    - audio (np.ndarray): Loaded audio data.
    - sr (int): Sampling rate of the loaded audio.
    """
    try:
        audio, sr = librosa.load(filepath, sr=sr)
        return audio, sr
    except Exception as e:
        print(f"Error reading audio file {filepath}: {e}")
        return None, None
```

Function: normalize_audio(audio)

- **Purpose**: Normalizes the amplitude of audio data to ensure consistency across different recordings.
- **Why Use**: Helps in standardizing audio inputs before applying further processing steps.
- **Code**:

```python
def normalize_audio(self, audio):
    """
    Normalize audio data to a standard amplitude range.

    Parameters:
    - audio (np.ndarray): Audio data.

    Returns:
    - audio_normalized (np.ndarray): Normalized audio data.
    """
    max_amp = np.max(np.abs(audio))
    if max_amp > 0:
        audio_normalized = audio / max_amp
    else:
        audio_normalized = audio
    return audio_normalized
```

Function:resample_audio(audio,orig_sr, target_sr=16000)

- **Purpose**: Adjusts the sampling rate of audio data to a specified target rate.
- **Why Use**: Ensures uniformity in sampling rates across all audio files, which is crucial for downstream analysis.
- **Code**:

```python
def resample_audio(self, audio, orig_sr, target_sr=16000):
    """
    Resample audio to a target sampling rate.

    Parameters:
    - audio (np.ndarray): Audio data.
    - orig_sr (int): Original sampling rate.
    - target_sr (int): Target sampling rate (default: 16000).

    Returns:
    - resampled_audio (np.ndarray): Resampled audio data.
    """
    resampled_audio = librosa.resample(audio, orig_sr=orig_sr, target_sr=target_sr)
    return resampled_audio
```

Function:pre_emphasis(audio, pre_emphasis_cof=0.97)

- **Purpose**: Applies a pre-emphasis filter to enhance high-frequency components in audio signals.
- **Why Use**: Improves the signal-to-noise ratio and assists in noise reduction.
- **Code**:

```
def pre_emphasis(self, audio, pre_emphasis_coef=0.97):
    """
    Apply pre-emphasis filter to the audio.

    Parameters:
    - audio (np.ndarray): Audio data.
    - pre_emphasis_coef (float): Pre-emphasis coefficient (default: 0.97).

    Returns:
    - emphasized_audio (np.ndarray): Audio data with pre-emphasis applied.
    """
    emphasized_audio = np.append(audio[0], audio[1:] - pre_emphasis_coef * audio[:-1])
    return emphasized_audio

def reduce_noise(self, audio, sr):
```

Function: reduce_noise(audio, sr)

- **Purpose**: Utilizes the noisereduce library to reduce noise from audio data.
- **Why Use**: Enhances the clarity of speech signals by mitigating background noise.
- **Code**:

```
def reduce_noise(self, audio, sr):
    """
    Reduce noise in audio using the noisereduce library.

    Parameters:
    - audio (np.ndarray): Audio data.
    - sr (int): Sampling rate of the audio.

    Returns:
    - reduced_noise_audio (np.ndarray): Noise-reduced audio data.
    """
    reduced_noise_audio = nr.reduce_noise(y=audio, sr=sr)
    return reduced_noise_audio
```

Function:process_all_files(output_dir,

target_sr=16000, pre_emphasis_coef=0.97)

- **Purpose**: Iteratively processes all audio files listed in the dataset with the defined preprocessing steps and saves the processed files to a specified output directory.
- **Why Use**: Automates the preprocessing pipeline for large datasets, ensuring consistency and efficiency.
- **Code**:

```python
def process_all_files(self, output_dir, target_sr=16000, pre_emphasis_coef=0.97):
    """
    Process all audio files listed in the dataset with various preprocessing steps and save them to the specified out

    Parameters:
        - output_dir (str): Path to the directory where the processed audio files will be saved.
        - target_sr (int): Target sampling rate for resampling (default: 16000).
        - pre_emphasis_coef (float): Pre-emphasis coefficient (default: 0.97).

    This method iterates through each row in the dataset, extracts the audio file path, loads the file,
    applies preprocessing steps, and saves the processed file to the output directory.
    """
    for index, row in self.data.iterrows():
        audio_path = row['audio_path']
        output_path = os.path.join(output_dir, os.path.basename(audio_path))

        print(f"Processing file: {audio_path}")

        # Read audio file
        audio, sr = self.read_audio(audio_path)
        if audio is None:
            continue

        # Preprocessing steps
        audio = self.normalize_audio(audio)
        audio = self.resample_audio(audio, orig_sr=sr, target_sr=target_sr)
        audio = self.pre_emphasis(audio, pre_emphasis_coef=pre_emphasis_coef)
        audio = self.reduce_noise(audio, sr)

        # Save processed audio
        sf.write(output_path, audio, sr)
        # print(f"Processed audio saved at {output_path}")
```

Function: plot_waveform(audio_path)

- **Purpose**: Generates a visual representation of the audio waveform.
- **Why Use**: Helps in inspecting the temporal characteristics of audio signals, aiding in quality assessment and troubleshooting.
- **Code**:

```
    def plot_waveform(self, audio_path):
        """
        Plot the waveform of the audio file.

        Parameters:
        - audio_path (str): Path to the audio file.
        """
        # Load audio
        audio, sr = self.read_audio(audio_path)
        if audio is None:
            print(f"Failed to load audio from {audio_path}")
            return

        # Plot waveform
        plt.figure(figsize=(14, 5))
        librosa.display.waveshow(audio, sr=sr)
        plt.title('Waveform')
        plt.xlabel('Time')
        plt.ylabel('Amplitude')
        plt.show()
```

Example Usage

- Here is an example of how to use the AudioProcessor class with a sample dataset
- **Code:**

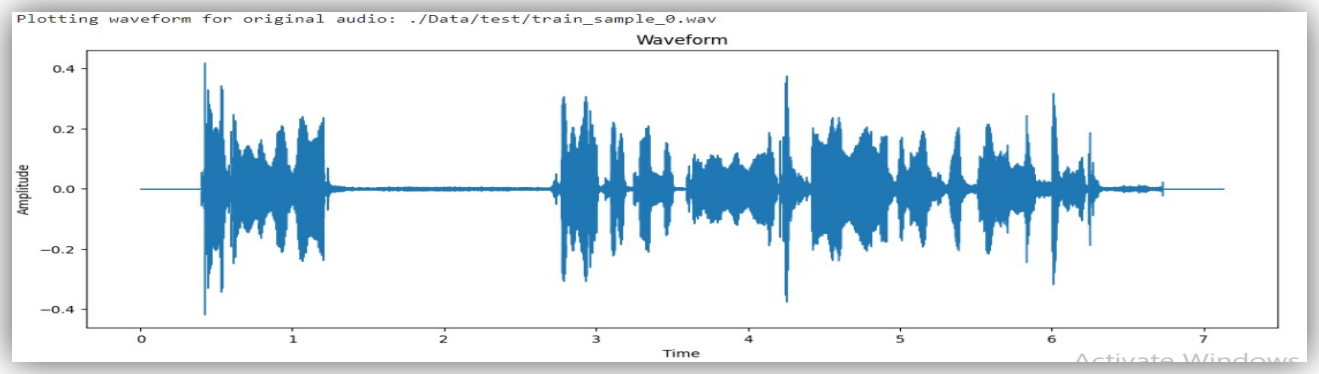
```
# Assuming train.csv is located in './Data/train.csv' and audio files are in './Data/test/'
train_df = pd.read_csv('./Data/train.csv')

# Update the audio paths to match the directory structure
train_audio_dir = './Data/test'
train_df['audio_path'] = train_df['audio'].apply(lambda x: os.path.join(train_audio_dir, x + '.wav'))

# Define the output directory for processed audio files
output_audio_dir = './Data/Clean_Data'

# Create the output directory if it doesn't exist
if not os.path.exists(output_audio_dir):
    os.makedirs(output_audio_dir)

# Instantiate the AudioProcessor with the training data
audio_processor = AudioProcessor(train_df)

# Process all files and save to the output directory
audio_processor.process_all_files(output_audio_dir)

# Plot waveforms before and after processing for the first audio file
original_audio_path = train_df['audio_path'][0]
processed_audio_path = os.path.join(output_audio_dir, os.path.basename(original_audio_path))

# Plot original audio waveform
print(f"Plotting waveform for original audio: {original_audio_path}")
audio_processor.plot_waveform(original_audio_path)

# Plot processed audio waveform
print(f"Plotting waveform for processed audio: {processed_audio_path}")
audio_processor.plot_waveform(processed_audio_path)
```

When the above code is executed, it will process the audio files and plot the waveforms for the original and processed audio. Here are sample outputs for the audio processing steps:

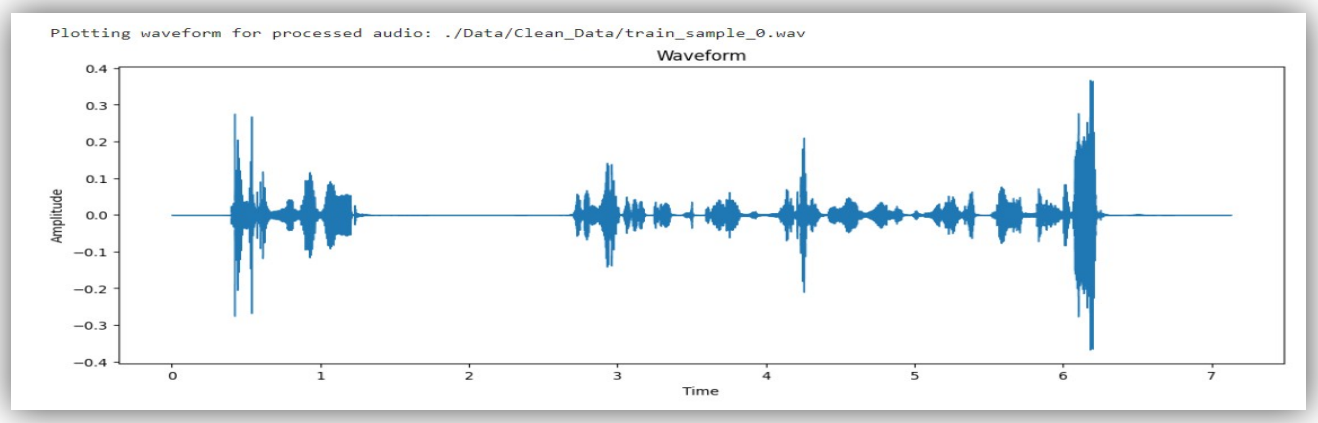
- **Processing files:** Messages indicating which files are being processed.
- **Plotting waveforms:** Visual plots of the original and processed audio waveforms

```
Processing file:  ./Data/test/train_sample_0.wav
Processing file:  ./Data/test/train_sample_1.wav
Processing file:  ./Data/test/train_sample_2.wav
Processing file:  ./Data/test/train_sample_3.wav
Processing file:  ./Data/test/train_sample_4.wav
Processing file:  ./Data/test/train_sample_5.wav
Processing file:  ./Data/test/train_sample_6.wav
```

Before Preprocessing



After Preprocessing



Data Augmentation

- Data augmentation is a technique used to increase the diversity of the training dataset without actually collecting new data. This is particularly useful in speech recognition projects, where obtaining large amounts of labeled audio data can be challenging. In this section, we demonstrate how to augment audio data by adding white noise and applying time-stretching.

Augmentation Functions

add_white_noise

- *This function adds white noise to the audio signal. Adding noise helps the model become more robust to variations and noise in real-world scenarios.*
- **Code:**

```
def add_white_noise(signal, noise_percentage_factor):
    noise = np.random.normal(0, signal.std(), signal.size)
    augmented_signal = signal + noise * noise_percentage_factor
    return augmented_signal
```

Explanation:

- **signal**: The input audio signal as a NumPy array.
- **noise_percentage_factor**: A float value representing the amount of noise to add.
- **noise**: Randomly generated noise based on the standard deviation of the input signal.
- **augmented_signal**: The original signal with added white noise.

Usage:

- This function is used to create augmented versions of the original audio by making it more resilient to noise.

time_stretch

- This function performs time-stretching on the audio signal, effectively speeding it up or slowing it down. Time-stretching helps the model handle variations in speech speed.

- **Code:**

```
def time_stretch(signal, time_stretch_rate):
    return librosa.effects.time_stretch(signal, rate=time_stretch_rate)
```

Explanation:

- **signal**: The input audio signal as a NumPy array.
- **time_stretch_rate**: A float value representing the stretch rate. Values greater than 1 speed up the signal, while values less than 1 slow it down.
- **time_stretched_signal**: The time-stretched version of the input signal.

Usage:

- This function is used to create augmented versions of the original audio by varying the speed of the audio playback.

Processing and Augmenting Audio Data

- This section reads the audio files listed in a CSV file, applies the augmentation functions, and saves the augmented audio files.
- **Code:**

```python
# Load the CSV file
csv_file = './Data/train.csv'
audio_directory = './Data/train/'
output_directory = './Data/train/'
df = pd.read_csv(csv_file)

# Ensure output directory exists
os.makedirs(output_directory, exist_ok=True)

# Prepare lists to hold new audio IDs and transcripts
new_audio_ids = []
new_transcripts = []

# Process each audio file
for index, row in df.iterrows():
    audio_id = row['audio']
    transcript = row['transcript']
    audio_path = os.path.join(audio_directory, f'{audio_id}.wav')

    if os.path.exists(audio_path):
        # Load the audio file
        signal, sr = librosa.load(audio_path, sr=None)

        # Perform augmentation
        augmented_signal_noise = add_white_noise(signal, noise_percentage_factor=0.05)
        augmented_signal_stretch9 = time_stretch(signal, time_stretch_rate=0.9)
        augmented_signal_stretch7 = time_stretch(signal, time_stretch_rate=0.7)

        # Save augmented audio files
        noise_audio_id = f'{audio_id}_noise'
        stretch_audio_id9 = f'{audio_id}_stretch9'
        stretch_audio_id7 = f'{audio_id}_stretch7'

        sf.write(os.path.join(output_directory, f'{noise_audio_id}.wav'), augmented_signal_noise, sr)
        sf.write(os.path.join(output_directory, f'{stretch_audio_id9}.wav'), augmented_signal_stretch9, sr)
        sf.write(os.path.join(output_directory, f'{stretch_audio_id7}.wav'), augmented_signal_stretch7, sr)

        # Add new audio IDs and transcripts to the lists
        new_audio_ids.extend([audio_id, noise_audio_id, stretch_audio_id9, stretch_audio_id7])
        new_transcripts.extend([transcript, transcript, transcript, transcript])

# Create new DataFrame and save to CSV
augmented_df = pd.DataFrame({'audio': new_audio_ids, 'transcript': new_transcripts})
augmented_csv_file = os.path.join(output_directory, 'augmented_data.csv')
augmented_df.to_csv(augmented_csv_file, index=False)

print(f'Augmented data saved to {augmented_csv_file}')
```

Explanation:

- **Load the CSV file**: Reads the CSV file containing the list of audio files and their transcripts.
- **Ensure output directory exists**: Creates the output directory if it doesn't already exist.
- **Prepare lists**: Initializes lists to hold the IDs and transcripts of the augmented audio files.
- **Process each audio file**: Iterates over each row in the CSV file, loads the audio file, and applies the augmentation functions.

- **Save augmented audio files**: Writes the augmented audio signals to new files and adds their IDs and transcripts to the lists.
- **Create new DataFrame**: Creates a new DataFrame containing the IDs and transcripts of the original and augmented audio files and saves it to a new CSV file.

Usage:

- This script is used to systematically augment all audio files in the dataset, creating a more diverse training set for the model.

Plotting Waveforms and STFT Features

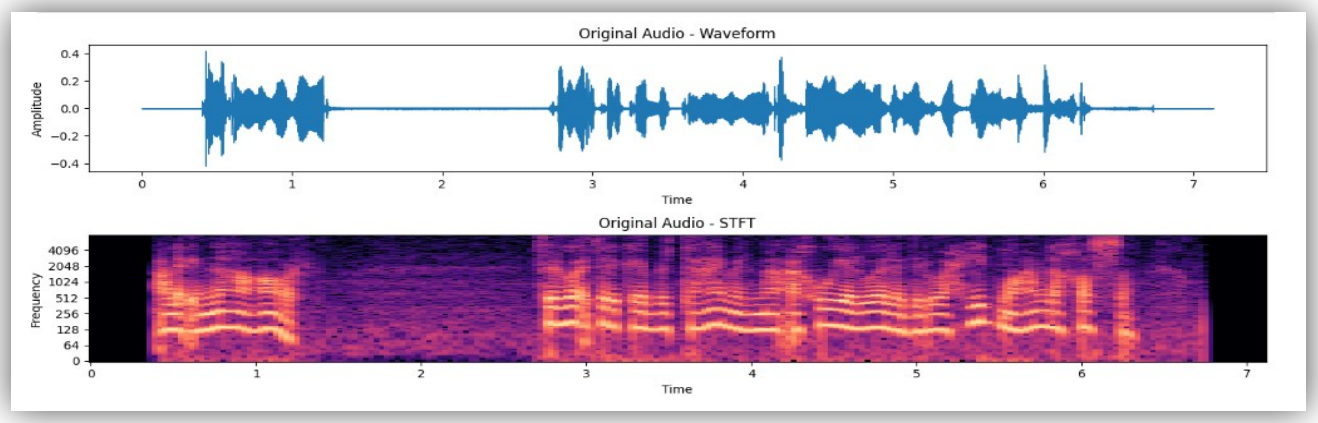
- This segment plots the waveforms and Short-Time Fourier Transform (STFT) features of the original and augmented audio samples, providing a visual comparison.
- **Code:**

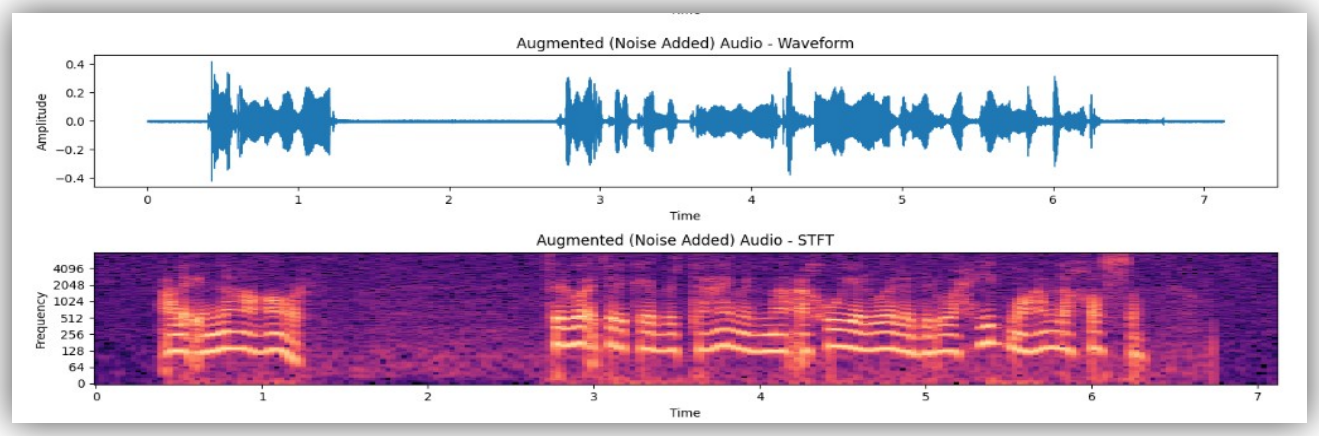
```
def plot_waveform_stft(audio_path, title):
    # Load audio
    signal, sr = librosa.load(audio_path, sr=None)

    # Plot waveform
    plt.figure(figsize=(14, 5))
    plt.subplot(2, 1, 1)
    librosa.display.waveshow(signal, sr=sr)
    plt.title(f'{title} - Waveform')
    plt.xlabel('Time')
    plt.ylabel('Amplitude')

    # Plot STFT
    plt.subplot(2, 1, 2)
    D = librosa.amplitude_to_db(np.abs(librosa.stft(signal)), ref=np.max)
    librosa.display.specshow(D, sr=sr, x_axis='time', y_axis='log')
    plt.title(f'{title} - STFT')
    plt.xlabel('Time')
    plt.ylabel('Frequency')

    plt.tight_layout()
    plt.show()

# Plot for original audio
plot_waveform_stft(original_audio_path, 'Original Audio')

# Plot for augmented noise-added audio
plot_waveform_stft(noise_audio_path, 'Augmented (Noise Added) Audio')

# Plot for augmented time-stretched (rate=0.9) audio
plot_waveform_stft(stretch9_audio_path, 'Augmented (Time Stretch 0.9) Audio')

# Plot for augmented time-stretched (rate=0.7) audio
plot_waveform_stft(stretch7_audio_path, 'Augmented (Time Stretch 0.7) Audio')
```

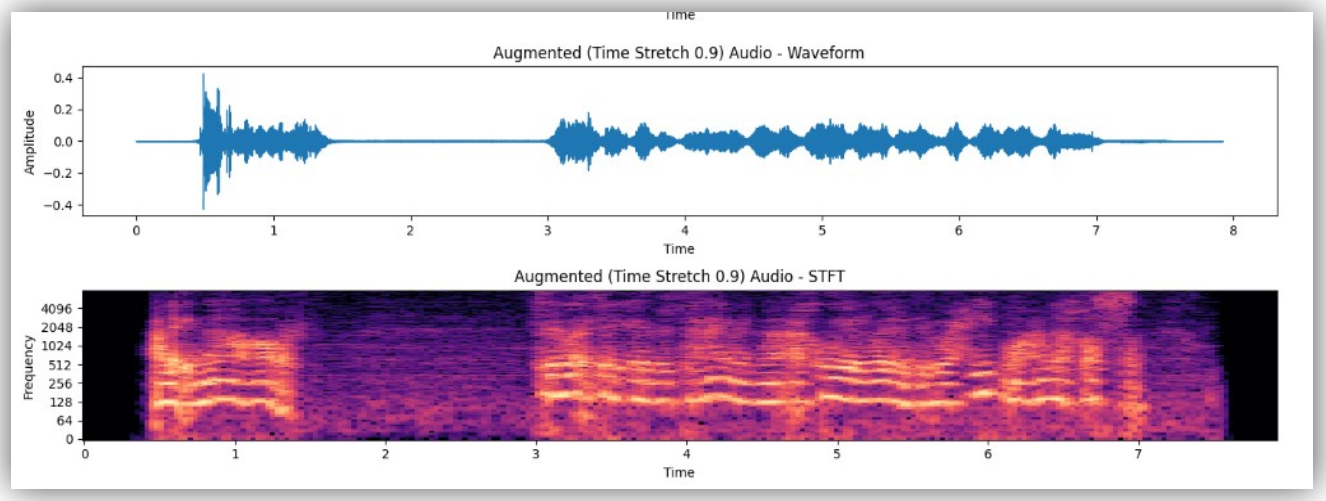
Explanation:

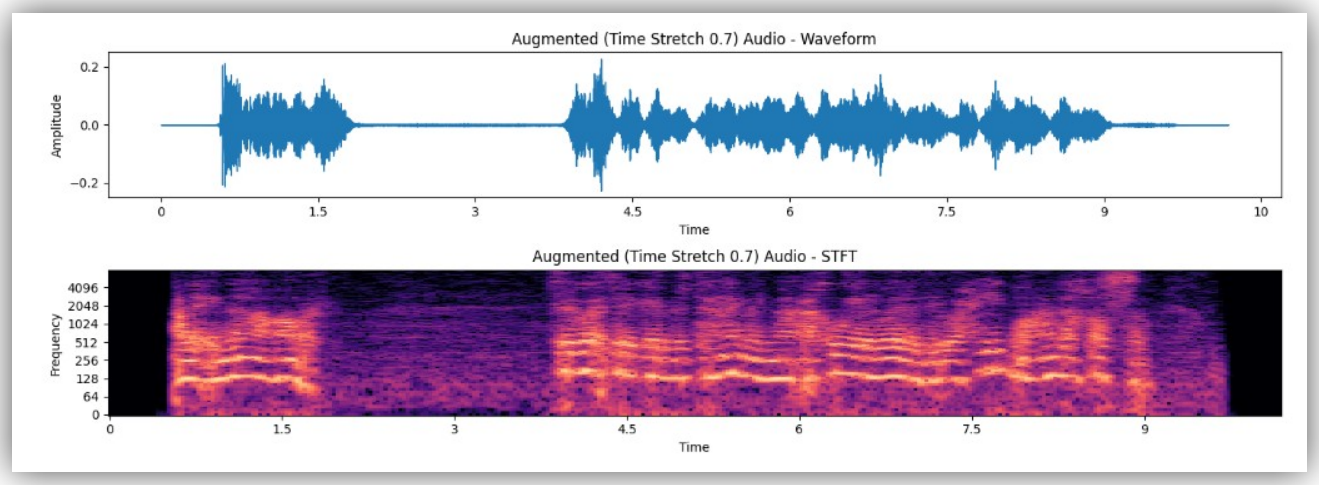
- **plot_waveform_stft**: A function to plot both the waveform and the STFT of an audio file, providing a visual representation of the audio signal.
- **original_audio_id, noise_audio_id, stretch_audio_id9, stretch_audio_id7**: IDs of the original and augmented audio files.
- **Load audio files**: Loads the original and augmented audio files for plotting.
- **Plot for original audio**: Plots the waveform and STFT for the original audio file.
- **Plot for augmented noise-added audio**: Plots the waveform and STFT for the audio file with added white noise.
- **Plot for augmented time-stretched (rate=0.9) audio**: Plots the waveform and STFT for the time-stretched audio

Output






Time
Augmented (Time Stretch 0.9) Audio - Waveform
Amplitude
0.4
0.2
0.0
−0.2
−0.4
0
1
2
3
4
5
6
7
8
Time
Augmented (Time Stretch 0.9) Audio - STFT
Frequency
4096
2048
1024
512
256
128
64
0
0
1
2
3
4
5
6
7
Time


Augmented (Time Stretch 0.7) Audio - Waveform
Amplitude
0.2
0.0
−0.2
0
1.5
3
4.5
6
7.5
9
10
Time
Augmented (Time Stretch 0.7) Audio - STFT
Frequency
4096
2048
1024
512
256
128
64
0
0
1.5
3
4.5
6
7.5
9
Time

3
main.ipynb

- **This notebook aims to load and preprocess audio datasets for raining a machine learning model for sound recognition.**
- **The librosa library is used to analyze audio and extract spectral features, enabling us to generate spectrogram plots for training purposes.**
- **The notebook also includes processes for noise reduction and improving audio data quality to enhance model performance. Subsequently,**
- **the data is prepared and split into training and testing sets to effectively evaluate the model's performance.**

Load Data with Extract SFTF Features

- This code loads, preprocesses, and splits audio data into
  training and testing sets, performing necessary operations to
  extract spectrograms and verify their validity

Load Data

- **Purpose:** The primary purpose of using load_data is to load training and testing data from CSV files and given audio
  directories (train_csv and train_audio_dir for training, adapt_csv and adapt_audio_dir for adaptation), and prepare them for model training.
- **Why use:** The load_data function simplifies the process of loading and preparing audio data for model training,

thereby improving the efficiency and accuracy of machine learning applications in sound recognition.

- **Code:**

```
from utils import load_data

# Now, calling the load_data function
train_csv = "./Data/train.csv"  # Replace with the actual path to your CSV file
train_audio_dir = "./Data/train/"  # Replace with the directory containing your audio files

adapt_csv = "./Data/adapt.csv"  # Replace with the actual path to your CSV file
adapt_audio_dir = "./Data/Clean_Data/"  # Replace with the directory containing your audio files

train_dataset, validation_dataset, df_train, df_val = load_data(train_csv, train_audio_dir, adapt_csv, adapt_audio_dir)
```

Extract Spectrogram

- **Purpose:** The goal of calculating the size of the training and validation datasets is to determine the number of items in each, helping to assess the amount of data available for training and validation
- **Why use:** This calculation is used to ensure proper data distribution between training and validation sets, ensuring that the model receives sufficient data to learn effectively without conflicting data used in training and evaluation.

- **Code:**

```
# Calculate the size of the validation dataset
validation_size = sum(1 for _ in validation_dataset.unbatch())

print(f"Size of the training set: {train_size}")
print(f"Size of the validation set: {validation_size}")
```

Printing the spectrogram and label shapes

- **Purpose:** The purpose of printing the spectrogram and label shapes is to understand the structure and format of the data that will be used for training.
  Why use: This is used to verify the correctness of the spectrogram extraction and data preparation process, ensuring that the data is in the correct format and

suitable for
training

- **Code:**

```
for spectrogram, label in train_dataset.take(1):  # Example of taking one batch
    print("Spectrogram shape:", spectrogram.shape)
    print("Label shape:", label.shape)
```

Output

```
Size of the training set: 24
Size of the validation set: 7
Spectrogram shape: (24, 883, 193)
Label shape: (24, 105)
```

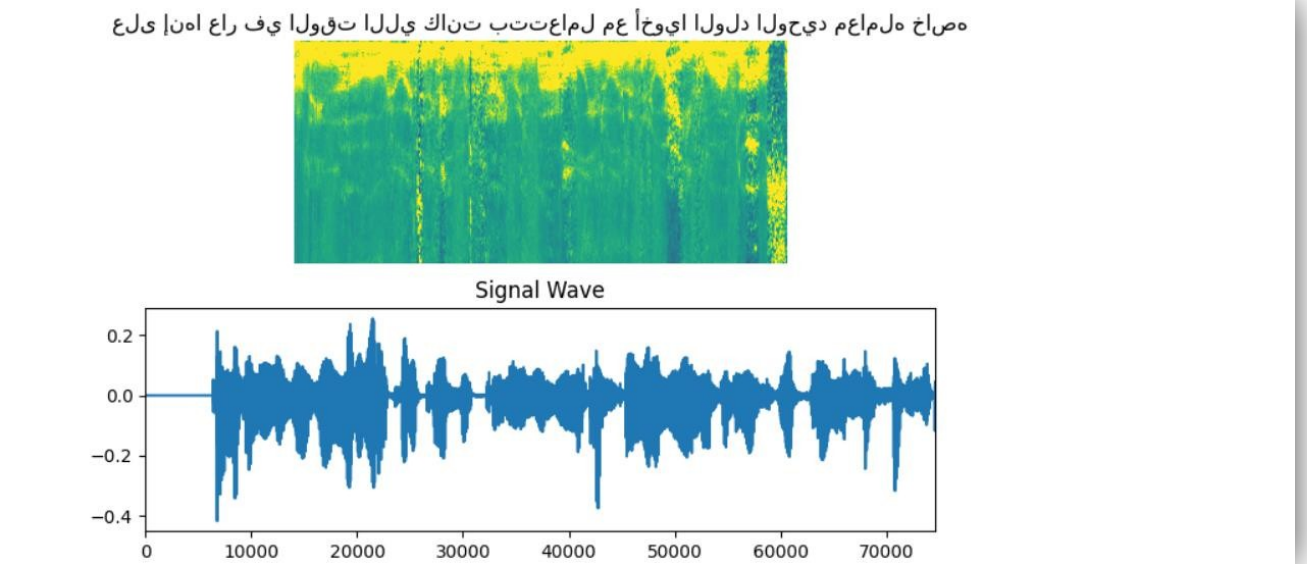
Sample From Training Dataset

- **Purpose:** The goal of this code is to visualize and understand the audio data from the training set, ensure the correctness of the spectrogram extraction, and analyze the audio signal to verify the quality of the data used in training

- **Code:**

```
from features_extraction import num_to_char

fig = plt.figure(figsize=(8, 5))
for batch in train_dataset.take(1):
    spectrogram = batch[0][0].numpy()
    spectrogram = np.array([np.trim_zeros(x) for x in np.transpose(spectrogram)])
    label = batch[1][0]
    # Spectrogram
    label = tf.strings.reduce_join(num_to_char(label)).numpy().decode("utf-8")
    ax = plt.subplot(2, 1, 1)
    ax.imshow(spectrogram, vmax=1)
    ax.set_title(label)
    ax.axis("off")
    # Wav
    file = tf.io.read_file(df_train["audio_path"][0])
    audio, _ = tf.audio.decode_wav(file)
    audio = audio.numpy()
    ax = plt.subplot(2, 1, 2)
    plt.plot(audio)
    ax.set_title("Signal Wave")
    ax.set_xlim(0, len(audio))
    display.display(display.Audio(np.transpose(audio), rate=16000))
plt.show()
```

Output



Load Model and Predict with It

Custom CTC Loss Function

- **Purpose:** This function defines a custom loss function for the Connectionist Temporal Classification (CTC) used in training models for sequence-to-sequence tasks, such as speech recognition.
- **Code:**

```python
@register_keras_serializable()
def CTCLoss(y_true, y_pred):
    batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
    input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
    label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")

    input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")
    label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")

    loss = keras.backend.ctc_batch_cost(y_true, y_pred, input_length, label_length)
    return loss
```

Explanation:

- **@register_keras_serializable():** This decorator allows the custom function to be serialized and deserialized as part of the Keras model, making it possible to save and load models that use this loss function.

- **CTCLoss(y_true, y_pred):** The function takes two arguments: y_true (the true labels) and y_pred (the predicted labels).
- **batch_len, input_length, label_length:** These variables are calculated to determine the lengths of the batch, input sequence, and label sequence, respectively.
- **loss = keras.backend.ctc_batch_cost(y_true, y_pred, input_length, label_length):** The actual CTC loss is computed using the Keras backend function ctc_batch_cost, which takes into account the true labels, predicted labels, and their respective lengths.

Load Model Configuration and Weights

- **Purpose:** This section loads a pre-trained model's configuration and weights, preparing it for making predictions.
- **Code:**

```
# Load model configuration from JSON file
with open('./Models/model_config.json', 'r') as json_file:
    model_config = json.load(json_file)

model = keras.models.model_from_json(json.dumps(model_config))
```

```
# Load the model weights
model.load_weights('./Models/Model.h5')
```

Explanation:

- **with open('./Models/model_config.json', 'r') as json_file:** Open the JSON file containing the model configuration.
- **model_config = json.load(json_file):** Load the JSON configuration into a Python dictionary.
- **keras.models.model_from_json(json.dumps(model_config)):** Create a Keras model from the JSON configuration.
- **model.load_weights('./Models/Model.h5'):** Load the pre-trained weights into the model.

Compile the Model

- **Purpose:** To compile the model using the Adam optimizer and the custom CTC loss function, making it ready for evaluation or prediction.
- **Code:**

```
# Compile the model with custom CTC loss function
model.compile(optimizer=keras.optimizers.Adam(learning_rate=1e-6), loss=CTCLoss)

model.summary()
```

Explanation:

- **model.compile(optimizer=keras.optimizers.Adam(learning_rate =1e-6), loss=CTCLoss):** The model is compiled with the Adam optimizer (learning rate set to 1e-6) and the custom CTC loss function.
- **model.summary():** Print a summary of the model's architecture, including the layers, their output shapes, and the number of parameters.

Output

Model: "DeepSpeech_2"

Layer (type)	Output Shape	Param #
input (InputLayer)	(None, None, 193)	0
expand_dim (Reshape)	(None, None, 193, 1)	0
conv_1 (Conv2D)	(None, None, 97, 32)	14,432
conv_1_bn (BatchNormalization)	(None, None, 97, 32)	128
conv_1_relu (ReLU)	(None, None, 97, 32)	0
conv_2 (Conv2D)	(None, None, 49, 32)	236,544
conv_2_bn (BatchNormalization)	(None, None, 49, 32)	128
conv_2_relu (ReLU)	(None, None, 49, 32)	0
reshape (Reshape)	(None, None, 1568)	0
bidirectional_1 (Bidirectional)	(None, None, 1024)	6,395,904
dropout (Dropout)	(None, None, 1024)	0
bidirectional_2 (Bidirectional)	(None, None, 1024)	4,724,736
dropout_1 (Dropout)	(None, None, 1024)	0
bidirectional_3 (Bidirectional)	(None, None, 1024)	4,724,736
dropout_2 (Dropout)	(None, None, 1024)	0
bidirectional_4 (Bidirectional)	(None, None, 1024)	4,724,736
dropout_3 (Dropout)	(None, None, 1024)	0
bidirectional_5 (Bidirectional)	(None, None, 1024)	4,724,736
dense_1 (Dense)	(None, None, 1024)	1,049,600
dense_1_relu (ReLU)	(None, None, 1024)	0
dropout_4 (Dropout)	(None, None, 1024)	0
dense (Dense)	(None, None, 39)	39,975

Total params: 26,635,655 (101.61 MB)
Trainable params: 26,635,527 (101.61 MB)
Non-trainable params: 128 (512.00 B)

Test and Predict Transcriptions for the Test Data

- **Purpose:** This script processes test audio files, makes predictions using the trained model, and saves the predictions to a CSV file
- **Code:**

```python
from metrics import decode_batch_predictions
from features_extraction import preprocess_audio

# Define the directory containing your test audio files
test_audio_dir = './Data/test'
output_csv_path = './Data/predictions.csv'

# Get a list of test audio files
test_files = [os.path.join(test_audio_dir, f) for f in os.listdir(test_audio_dir) if f.endswith('.wav')]

# Initialize a list to store predictions
predictions = []

# Process each test file and make predictions
for audio_file in test_files:
    # Preprocess the audio file
    spectrogram = preprocess_audio(audio_file)
    spectrogram = tf.expand_dims(spectrogram, axis=0)  # Add batch dimension
    # Make prediction
    prediction = model.predict(spectrogram)
    # Decode prediction to text
    decoded_text = decode_batch_predictions(prediction)
    # Append the result with filename instead of full path
    predictions.append({'audio': os.path.splitext(os.path.basename(audio_file))[0], 'transcript': decoded_text[0]})

# Save predictions to CSV
pred_df = pd.DataFrame(predictions)
pred_df.to_csv(output_csv_path, index=False, encoding='utf-8')

print(f"Predictions saved to {output_csv_path}")
```

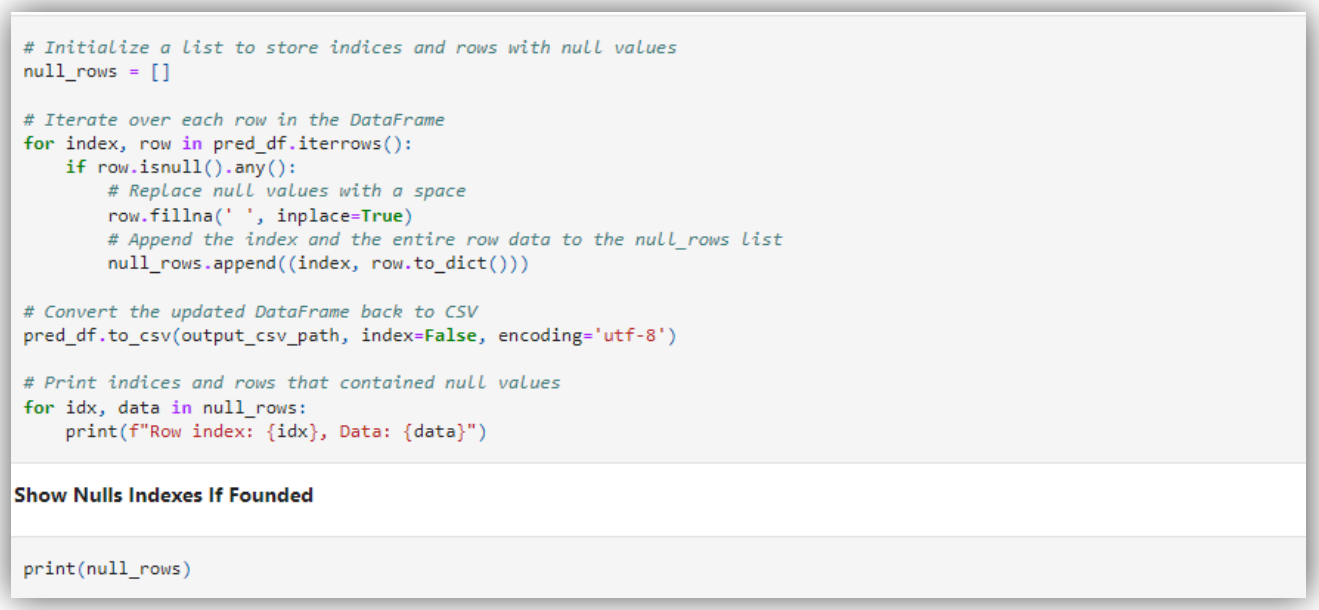
Explanation:

- **from metrics import decode_batch_predictions:** Import the function to decode batch predictions.
- **from features_extraction import preprocess_audio:** Import the function to preprocess audio files.
- **test_audio_dir, output_csv_path:** Define the directory for test audio files and the path for the output CSV file.
- **test_files:** Get a list of all .wav files in the test directory.
- **predictions:** Initialize an empty list to store the predictions.
- **for audio_file in test_files:** Loop through each test audio file.
    - **spectrogram = preprocess_audio(audio_file):** Preprocess the audio file to obtain its spectrogram.

    - **spectrogram = tf.expand_dims(spectrogram, axis=0):** Add a batch dimension to the spectrogram.
    - **prediction = model.predict(spectrogram):** Make a prediction using the trained model.
    - **decoded_text = decode_batch_predictions(prediction):** Decode the prediction to text.
    - **predictions.append({'audio': os.path.splitext(os.path.basename(audio_file))[0], 'transcript': decoded_text[0]}):** Append the prediction to the list, storing the audio filename (without path or extension) and the transcript.
- **pred_df = pd.DataFrame(predictions):** Create a DataFrame from the predictions list.
- **pred_df.to_csv(output_csv_path, index=False, encoding='utf8'):** Save the predictions DataFrame to a CSV file.

Handling Null Values in Predictions

- **Purpose:** To handle any null values in the predictions by replacing them with spaces and logging the affected rows.
- **Code:**



```
# Initialize a list to store indices and rows with null values
null_rows = []

# Iterate over each row in the DataFrame
for index, row in pred_df.iterrows():
    if row.isnull().any():
        # Replace null values with a space
        row.fillna(' ', inplace=True)
        # Append the index and the entire row data to the null_rows list
        null_rows.append((index, row.to_dict()))

# Convert the updated DataFrame back to CSV
pred_df.to_csv(output_csv_path, index=False, encoding='utf-8')

# Print indices and rows that contained null values
for idx, data in null_rows:
    print(f"Row index: {idx}, Data: {data}")
```

Show Nulls Indexes If Founded

```
print(null_rows)
```

Explanation:

- **null_rows:** Initialize an empty list to store indices and rows with null values.
- **for index, row in pred_df.iterrows():** Iterate over each row in the DataFrame.
  - **if row.isnull().any():** Check if any values in the row are null.
  - **row.fillna(' ', inplace=True):** Replace null values with a space.
  - **null_rows.append((index, row.to_dict()):** Append the index and the entire row data to the null_rows list.
- **pred_df.to_csv(output_csv_path, index=False, encoding='utf-8'):** Save the updated DataFrame back to the CSV file.
- **for idx, data in null_rows:** Print the indices and rows that contained null values.
- **print(null_rows):** Show the list of null indices, if any are found.

conclusion

This project successfully implemented a speech recognition system using the DeepSpeech 2 model. The project demonstrates feature extraction, model training, and evaluation. By combining these components, the system can convert spoken audio into text.
All parts of the project except for some small functions are built from scratch
In the end, we tell you that any success is from Allah , and whatever oversight, forgetfulness, or something unsuccessful is from us and from Eblis.

Some of the challenges we face and how we overcame them

During our project, we faced many important challenges. Initially, we faced limitations in our computational resources, which were not sufficient for our workload. To address this issue, we invested in a Colab Pro subscription, providing us with more powerful and reliable computing resources.

In addition, after performing data augmentation, we did not observe any positive impact on the model performance during training. Hence, we decided to go back to training the model on the original dataset at its initial size, which helped us maintain the expected performance levels.

Finally, tight deadlines as well as scheduling conflicts with college exams required us to work around the clock, often without sleep or rest. Despite these difficulties,

we managed our time efficiently and worked in shifts whenever possible, ensuring continuous progress while trying to avoid burnout.

And Also that is the repo URL of Get Prediction And It Is Puplic : https://github.com/MO7AMED3TWAN/MTC-AIC2

Thanks All, My Beautiful Team Members and all supporters around the way of this competition