Transcription Model for Marathi Language

This code generates transcription of marathi audio files based on given audio files, using the openai/whisper-large-v3 deep learning model.

Install and Import necessary libraries

Installing and importing the required libraries,

```
! pip3 install accelerate

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

!pip install tensorflow
!pip install transformers

pip install tensorflow==2.12.0 tensorflow-probability==0.15.0

pip install datasets

import torch
from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor, pipeline
from datasets import load_dataset

!pip install jiwer

import jiwer
import re
import os
```

Load large language model

We are using openai/whisper-large-v3 model because its large-scale architecture and extensive pre-training on diverse and massive datasets. Its deep neural network design enables it to capture intricate patterns in spoken language, allowing for accurate and contextually rich transcriptions Its ability to understand and generate text from various accents, languages, and speech makes it versatile for diverse applications.

Loads the large language model for processing audio files

```
model_id = "openai/whisper-large-v3"

model = AutoModelForSpeechSeq2Seq.from_pretrained(
    model_id, torch_dtype=torch_dtype, use_safetensors=True
)

model.to(device)
```

Define function

Define a function that takes audio files as input and generate there transcriptions.

```
def get_transcription(filename: str):

transcription = model_output

return transcription
```

Define transcription generation function

Defines a function that generate transcriptions of audio files.

```
def get_transcription(filename: str):



    file_path = os.path.join(folder_path, filename)

    result = pipe(file_path, generate_kwargs={"language": "marathi"})



    return f"{filename} {result['text']}"


def process_and_save_transcriptions(folder_path: str, batch_size: int, output_file_path: str):


    all_results = []
    audio_files = [filename for filename in os.listdir(folder_path) if filename.endswith('.wav')]


    for i in range(0, min(len(audio_files), batch_size), batch_size):

        batch_files = audio_files[i:i+batch_size]


        for filename in batch_files:
            # Get transcription for the current file
            transcription = get_transcription(filename)


            all_results.append(transcription)


    with open(output_file_path, 'w', encoding='utf-8') as output_file:
        output_file.write('\n'.join(all_results))

    print(f'Processed {len(all_results)} files. Results saved to: {output_file_path}')


folder_path = '/content/drive/MyDrive/common_voice_test'
batch_size = 1816
output_file_path = '/content/drive/MyDrive/common_voice_text_results(8).txt'

process_and_save_transcriptions(folder_path, batch_size, output_file_path)
```

Download the file

```
file_id = '/content/drive/MyDrive'  # Replace with the actual file ID from the Google Drive link
gdown.download(f'https://drive.google.com/uc?id={file_id}', output_file_path + '_downloaded.txt')

print(f'Text file downloaded to: {output_file_path}_downloaded.txt')
```

Sort the trans file and the generated voice files

```
from google.colab import files

uploaded = files.upload()

with open('/content/trans.txt', 'r', encoding='utf-8') as file:
    lines = file.readlines()


sorted_lines = sorted(lines, key=lambda x: int(x.split('_')[-1].split('.')[0]))


with open('sorted_trans(1).txt', 'w', encoding='utf-8') as file:
    file.writelines(sorted_lines)
files.download('sorted_trans(1).txt')


from google.colab import files

uploaded = files.upload()
with open('/content/common_voice_text_results(8).txt', 'r', encoding='utf-8') as file:
    lines = file.readlines()

sorted_lines = sorted(lines, key=lambda x: int(''.join(filter(str.isdigit, x.split('_')[-1].split('.')[0]))))

with open('sorted_common_voice(8).txt', 'w', encoding='utf-8') as file:
    file.writelines(sorted_lines)
# Download the sorted_trans.txt file
files.download('sorted_common_voice(8).txt')
```

Evaluation Metric

Word error rate is used as a metric to evaluate the accuracy of text generation model because it provides a comprehensive measure of the overall discrepancy between the recognized output and the ground truth reference. WER considers the total number of substitutions, insertions, and deletions needed to transform the recognized sentence into the reference sentence.

```
!pip install jiwer

import jiwer
import re

# Function to calculate Word Error Rate between two sentences
def calculate_wer(reference, hypothesis):
    try:
        return jiwer.wer(reference, hypothesis)
    except ValueError as e:
        print(f"Error calculating WER: {e}")
        return None


with open('/content/sorted_trans(1).txt', 'r', encoding='utf-8') as file1:
    trans_lines = file1.readlines()

with open('/content/sorted_common_voice(8).txt', 'r', encoding='utf-8') as file2:
    common_voice_lines = file2.readlines()


if len(trans_lines) != len(common_voice_lines):
    print("The number of sentences in the two files is different.")


min_length = min(len(trans_lines), len(common_voice_lines))
wer_results = []
for ref, hyp in zip(trans_lines[:min_length], common_voice_lines[:min_length]):
    reference = re.findall(r'\b\w+\b', ref)  # Extract words from the reference sentence
    hypothesis = re.findall(r'\b\w+\b', hyp)  # Extract words from the hypothesis sentence


    max_length = max(len(reference), len(hypothesis))
    reference += [''] * (max_length - len(reference))
    hypothesis += [''] * (max_length - len(hypothesis))
    wer = calculate_wer(reference, hypothesis)
    if wer is not None:
        wer_results.append(wer)


with open('wer_results.txt', 'w', encoding='utf-8') as wer_file:
    for wer in wer_results:
        wer_file.write(f'{wer}\n')


from google.colab import files
files.download('wer_results.txt')
```

Final Code:

```
# -*- coding: utf-8 -*-
"""Final_Done_Colab_Akaike_Assignment_code.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1QoiBrqcJjD3V4emWMgoPYbKJksvjXcGz
"""

! pip3 install accelerate

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

!pip install tensorflow
!pip install transformers

pip install tensorflow==2.12.0 tensorflow-probability==0.15.0

pip install datasets

import torch
from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor, pipeline
from datasets import load_dataset

device = "cuda:0" if torch.cuda.is_available() else "cpu"
```

```
torch_dtype = torch.float16 if torch.cuda.is_available() else torch.float32

model_id = "openai/whisper-large-v3"

model = AutoModelForSpeechSeq2Seq.from_pretrained(
    model_id, torch_dtype=torch_dtype, use_safetensors=True
)

model.to(device)

processor = AutoProcessor.from_pretrained(model_id)

pipe = pipeline(
    "automatic-speech-recognition",
    model=model,
    tokenizer=processor.tokenizer,
    feature_extractor=processor.feature_extractor,
    max_new_tokens=128,
    chunk_length_s=30,
    batch_size=16,
    return_timestamps=True,
    torch_dtype=torch_dtype,
    device=device,
)

sample = '/content/common_voice_mr_31671592.wav'
result = pipe(sample, generate_kwargs={"language": "marathi"})
print(result["text"])

import os
import gdown

import os

def get_transcription(filename: str):

    # Construct the full file path
    file_path = os.path.join(folder_path, filename)

    result = pipe(file_path, generate_kwargs={"language": "marathi"})

    return f"{filename} {result['text']}"

def process_and_save_transcriptions(folder_path: str, batch_size: int, output_file_path: str):
    all_results = []

    audio_files = [filename for filename in os.listdir(folder_path) if filename.endswith('.wav')]


    for i in range(0, min(len(audio_files), batch_size), batch_size):
        # Extract the current batch of filenames
        batch_files = audio_files[i:i+batch_size]

        for filename in batch_files:
            # Get transcription for the current file
            transcription = get_transcription(filename)


            all_results.append(transcription)

    with open(output_file_path, 'w', encoding='utf-8') as output_file:
        output_file.write('\n'.join(all_results))

    print(f'Processed {len(all_results)} files. Results saved to: {output_file_path}')

folder_path = '/content/drive/MyDrive/common_voice_test'
batch_size = 1816
output_file_path = '/content/drive/MyDrive/common_voice_text_results(8).txt'

process_and_save_transcriptions(folder_path, batch_size, output_file_path)


file_id = '/content/drive/MyDrive'  # Replace with the actual file ID from the Google Drive
link
gdown.download(f'https://drive.google.com/uc?id={file_id}', output_file_path +
'_downloaded.txt')

print(f'Text file downloaded to: {output_file_path}_downloaded.txt')

from google.colab import files

uploaded = files.upload()
with open('/content/trans.txt', 'r', encoding='utf-8') as file:
    lines = file.readlines()

sorted_lines = sorted(lines, key=lambda x: int(x.split('_')[-1].split('.')[0]))

with open('sorted_trans(1).txt', 'w', encoding='utf-8') as file:
    file.writelines(sorted_lines)
```

```
files.download('sorted_trans(1).txt')

from google.colab import files

uploaded = files.upload()
with open('/content/common_voice_text_results(8).txt', 'r', encoding='utf-8') as file:
    lines = file.readlines()

sorted_lines = sorted(lines, key=lambda x: int(''.join(filter(str.isdigit, x.split('_')[-
1].split('.')[0]))))

with open('sorted_common_voice(8).txt', 'w', encoding='utf-8') as file:
    file.writelines(sorted_lines)
# Download the sorted_trans.txt file
files.download('sorted_common_voice(8).txt')

!pip install jiwer

import jiwer
import re

# Function to calculate Word Error Rate between two sentences
def calculate_wer(reference, hypothesis):
    try:
        return jiwer.wer(reference, hypothesis)
    except ValueError as e:
        print(f"Error calculating WER: {e}")
        return None

with open('/content/sorted_trans(1).txt', 'r', encoding='utf-8') as file1:
    trans_lines = file1.readlines()

with open('/content/sorted_common_voice(8).txt', 'r', encoding='utf-8') as file2:
    common_voice_lines = file2.readlines()

if len(trans_lines) != len(common_voice_lines):
    print("Warning: The number of sentences in the two files is different.")

min_length = min(len(trans_lines), len(common_voice_lines))
wer_results = []
for ref, hyp in zip(trans_lines[:min_length], common_voice_lines[:min_length]):
    reference = re.findall(r'\b\w+\b', ref)  # Extract words from the reference sentence
    hypothesis = re.findall(r'\b\w+\b', hyp)  # Extract words from the hypothesis sentence
    max_length = max(len(reference), len(hypothesis))
    reference += [''] * (max_length - len(reference))
    hypothesis += [''] * (max_length - len(hypothesis))

    wer = calculate_wer(reference, hypothesis)
    if wer is not None:
        wer_results.append(wer)

with open('wer_results.txt', 'w', encoding='utf-8') as wer_file:
    for wer in wer_results:
        wer_file.write(f'{wer}\n')

from google.colab import files
files.download('wer_results.txt')
```

Conclusion:

The program takes audio files from the user and generate its transcriptions using open-ai/whisper-large-v3 model. It generates transcription of all the marathi audio files. This model is useful for speech to text generation tasks.