MODEL SPECIFIC TASKS

T. SAI RISHIK

December 28, 2023

1 TASKS

1. Given an audio file, the model should predict the transcription for the audio and also print the respective WER for each audio file.
2. Train the openai whisper model on 5 different languages
    - Tamil
    - Telugu
    - English (Indian Accent)
    - Bengali
    - Hindi

2 MODELTEST FILE

2.1 Introduction

This document provides documentation for the python script, which utilizes the Whisper ASR system to transcribe audio data from a given CSV file.

2.2 Dependencies

Ensure the following Python packages are installed:

- `jiwer`
- `pandas`
- `os`
- `torchaudio`
- `transformers` (Hugging Face Transformers library)

2.3 Setup

1. Load the pretrained Whisper model from Hugging Face model hub using the identifier `openai/whisper-small`.
2. Prepare a CSV file (`duptest.csv`) containing audio file names in the 'audio' column and corresponding sentences in the 'sentence' column.
3. Ensure audio files are present in the specified directory (`audio_files_dir`).

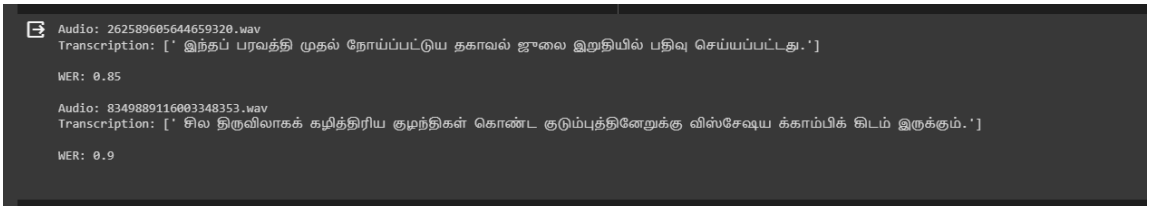

Figure 1: For the openai/whisper-small model

2.4 Usage

- The script processes each audio file, resamples it to a specified sampling rate, and generates transcriptions using the Whisper ASR model.
- The Word Error Rate (WER) is calculated using the `jiwer` library by comparing the predicted transcription with the actual sentence from the CSV file.
- WER values are stored in the `wer_list` for further analysis.
- The script prints the audio filename, transcriptions, and WER for each processed audio file.

2.5 Code Structure

The code is structured as follows:

- `WhisperProcessor` and `WhisperForConditionalGeneration` are initialized using the pre-trained Whisper model.
- Audio files are loaded, resampled, and converted into input features for the Whisper model.
- The Whisper model generates transcriptions, and WER is calculated and printed for each audio file.

2.6 Output

1. Check 1

3 CSVFileTrain FILE

3.1 Introduction

This document provides documentation for a Python script that performs fine-tuning and evaluation of the Whisper ASR model using the Hugging Face Transformers library.

3.2 Dependencies

Ensure the following Python packages are installed:

- `dataclasses`
- `typing`
- `torch`
- `datasets`
- `transformers`

3.3 Data Loading

The script loads Common Voice datasets from CSV files 'dup.csv' for training and 'duptest.csv' for testing) using the Hugging Face Datasets library.

3.4 Preprocessing

- The script uses a WhisperFeatureExtractor, WhisperTokenizer, and WhisperProcessor from the Hugging Face Transformers library to prepare the dataset.
- Audio files are downsampled from 49 kHz to 16 kHz for consistency.
- The `prepare_dataset` function is defined to load and preprocess audio data, compute log-Mel input features, and encode target text to label ids.

3.5 Data Collator

The script defines a `DataCollatorSpeechSeq2SeqWithPadding` class responsible for collating input features and labels. It pads inputs and labels to the maximum length and replaces padding with -100 to ignore loss correctly.

3.6 Model Configuration

The Whisper ASR model is configured using the Hugging Face Transformers library. The forced decoder ids and suppress tokens are set for fine-tuning.

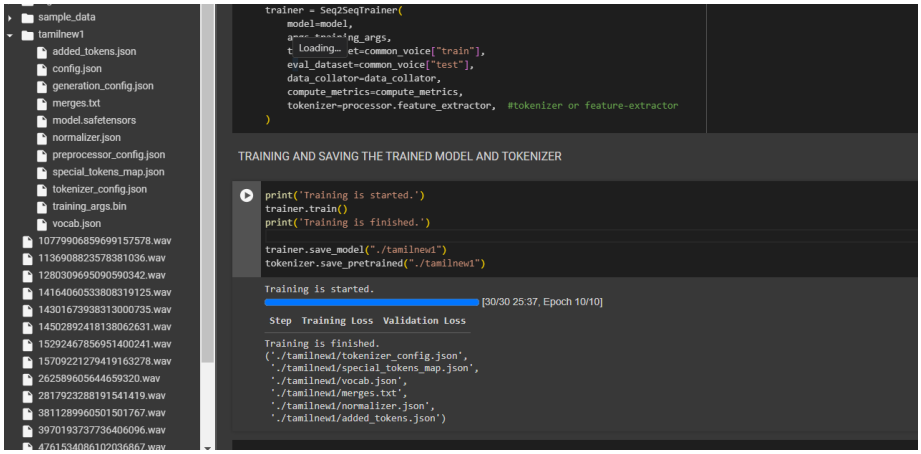

Figure 2: After the model completes its training

3.7 Training Configuration

- The script sets up training arguments using Seq2SeqTrainingArguments with options for evaluation strategy, batch sizes, save intervals, number of epochs, logging, and more.
- The compute metrics function calculates the Word Error Rate (WER) using the jiwer library.

3.8 Training

The script utilizes the Seq2SeqTrainer from the Hugging Face Transformers library to train the Whisper ASR model on the Common Voice dataset. The training progress is logged, and the model is saved.

3.9 Save Trained Model

The trained model and tokenizer are saved to the specified directory.

3.10 Output

1. Check Figure 2

4 Output after model is trained

- Use the file TrainedModelTest.py
- Check 3

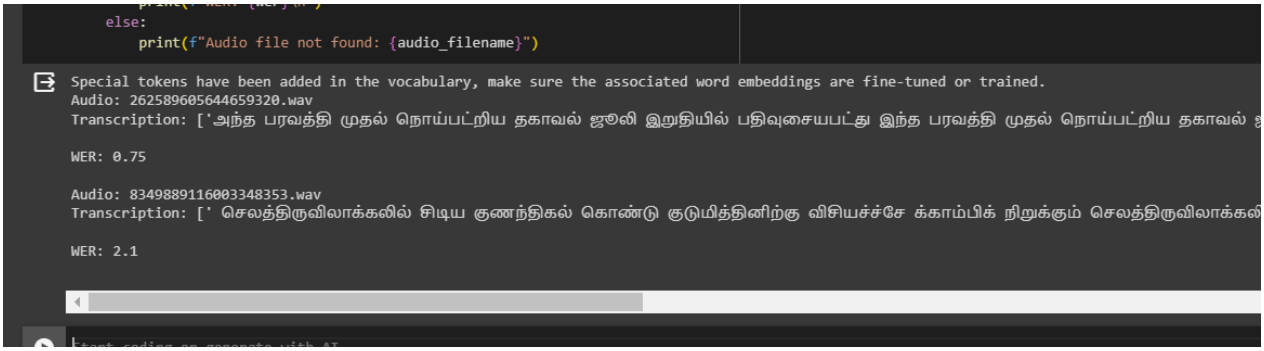

Figure 3: Result from trained model

5 PEFT

5.1 Error: WhisperForConditionalGeneration.forward() got an unexpected keyword argument 'input-ids'

- Check 4
- According to WhisperForConditionalGeneration documentation there is no argument input-id's. So, here using custom whisper model we set input-id's to None.
- Then an KeyError: input-features is generated

5.2 About KeyError: input-features

- Initially, common-voice is a dictionary with keys: train and test. Within train and test the keys are features and num-rows. Refer 5
- Then, in the prepare-dataset function we are printing every batch, each batch is a dictionary with keys as audio, sentence, input-features and labels. Refer 6
- Then common-voice["train"] is a dataset with keys as features and num-rows. Refer 7
- Then common-voice["train"].features is a dictionary with keys as input-features and labels. Refer 7
- Then common-voice["train].features["input-features"] is a sequence of sequence of float numbers. Refer 8
- Then at the end of data collator each batch is again printed out. The output is a list of dictionaries with keys as only label. Refer 9
- In that dictionaries there is no key names input-features

```python
from transformers import WhisperForConditionalGeneration, WhisperConfig

class CustomWhisperForConditionalGeneration(WhisperForConditionalGeneration):
    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        encoder_outputs=None,
        inputs_embeds=None,
        decoder_input_ids=None,
        decoder_attention_mask=None,
        head_mask=None,
        inputs_embeds_decoder=None,
        labels=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
        **kwargs
    ):
        # Your implementation of the forward method with the correct arguments
        # Make sure to call the base class forward method with the appropriate arguments
        return super().forward(
            input_ids=input_ids,
            attention_mask=attention_mask,
            encoder_outputs=encoder_outputs,
            inputs_embeds=inputs_embeds,
            decoder_input_ids=decoder_input_ids,
            decoder_attention_mask=decoder_attention_mask,
            head_mask=head_mask,
            inputs_embeds_decoder=inputs_embeds_decoder,
            labels=labels,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
            **kwargs
        )
```

Figure 4: Custom Whisper Model

```
DatasetDict({
    train: Dataset({
        features: ['audio', 'sentence'],
        num_rows: 5
    })
    test: Dataset({
        features: ['audio', 'sentence'],
        num_rows: 2
    })
})
```

Figure 5: common-voice

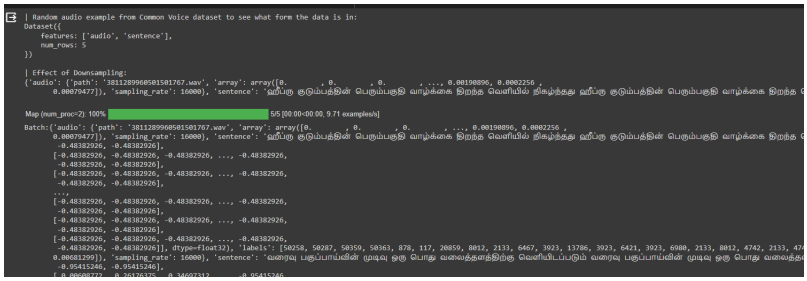

Figure 6: prepare data function(batch)

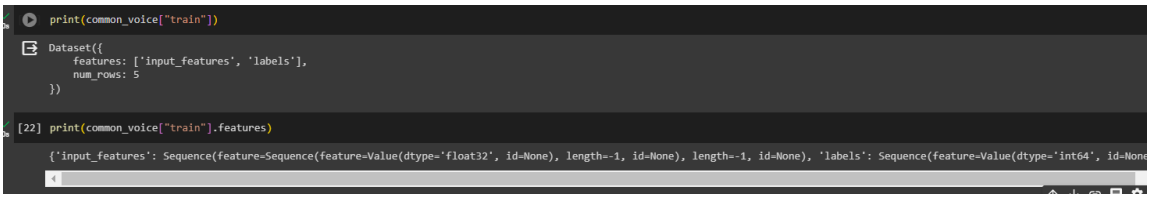

Figure 7: common-voice[”train”]

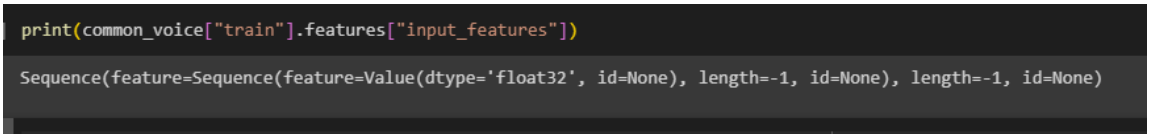

Figure 8: common-voice[”train].features[”input-features”]

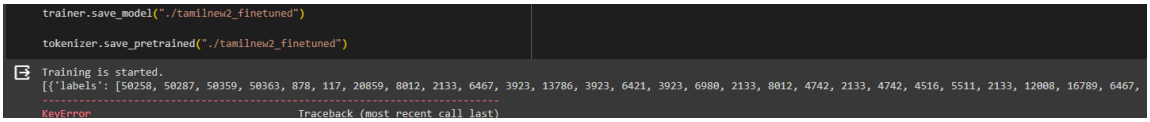

Figure 9: Data Collator batch