Speech to text

Audio to text translation

[Github Link]

Maor Ovadia

Dor Redlich

Omer Kalif

Gofna Ivry

Table of Contents

1. Introduction

1.1 Deep-learning and S2T

Deep Learning has changed the game in speech recognition with the introduction of end-to-end models. These models take in audio, and directly output transcriptions.

Both Deep Speech and LAS, are recurrent neural network (RNN) based architectures with different approaches to modeling speech recognition. Deep Speech uses the Connectionist Temporal Classification (CTC) loss function to predict the speech transcript. LAS uses a sequence-to-sequence network architecture for its predictions.

1.2 Our S2T project

This project is an Implementation of speech2text with the addition of a web user interface that allow you to upload an audio file and get the text translation of it by using a trained model.

Features

- Train speech2text.
- Language model support using kenlm.
- Easy start/stop capabilities in the event of crash or hard stop during training.
- Tensorboard support for visualizing training graphs.
- Simple web user interface to see the trained model's results.

2. Installation

2.1 Environment and versions

The project developed in PyCharm IDE on Ubuntu.

Python – version 3.9

Ubuntu – version 20.04

2.2 dependencies

Several libraries are needed to be installed for training to work:

Install PyTorch:

```
Pip3 install torch
```

Install Librosa:

```
Pip3 install librosa
```

For tensorboard training visualization, install tensorboardX.

This is optional, but recommended:

```
Pip3 install tensorboardx
```

For data preparation, download or install FFmpeg:

```
Pip3 install ffmpeg
```

Finally clone this repo and run this within the repo:

```
Pip3 install -r requirements.txt
```

3. Model Preparing

3.1 LibriSpeech dataset

Description: LibriSpeech is a corpus of approximately 1000 hours of read English speech with sampling rate of 16 kHz, prepared by Vassil Panayotov with the assistance of Daniel Povey. The data is derived from read audiobooks from the LibriVox project and has been carefully segmented and aligned.87

Download the dataset you want to use from openslr.

Run data/prepare_librispeech.py on the downloaded tar.gz file in terminal:

```
Python3 data/prepare_librispeech.py --zip_file *name_of_file*.tar.gz
--extracted_dir *name_of_extract_directory* --target_dir dataset --
manifest_path *name_of_file*.csv
```

3.2 custom datasets

To create a custom dataset, create a CSV file containing audio location and text pairs. This can be in the following format:

/path/to/audio.wav, transcription

/path/to/audio2.wav, transcription

...

Alternatively, create a Pandas Dataframe with the

columns filepath, text and save it using df.to_csv(path).

Note that only WAV files are supported.

If you use a sample rate other than 8K, specify it on the file

“train.py” here:

```
parser.add_argument('--train-manifest', help='path to train manifest csv', default=t
parser.add_argument('--val-manifest', help='path to validation manifest csv', defaul
parser.add_argument('--sample-rate', default=8000, type=int, help='Sample rate')
parser.add_argument('--window-size', default=.02, type=float, help='Window size for
parser.add_argument('--window-stride', default=.01, type=float, help='Window sstride
```

3.3 Training

To train model you need to pay attention to few parameters in

“train.py” file.

1. Epochs- The number of epochs. it is a hyperparameter that defines the number of times that the learning algorithm will work through the entire training dataset.

One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters.

2. Continue_from – If you interest to train an exist model this parameter should be ‘true’, and if you interest to train a new model, this parameter should be ‘False’.
3. Continue_from_path- the path of the exist model you want to continue its training.
4. Path to the data set for the train and validation:

```
train_manifest = 'train-clean-100.csv'
validation_manifest = 'train-clean-100.csv'
```

These and other parameters like-

Window size of spectrogram, Batch size, Directory to save the models and more,

can be viewed and modified in the same file here:

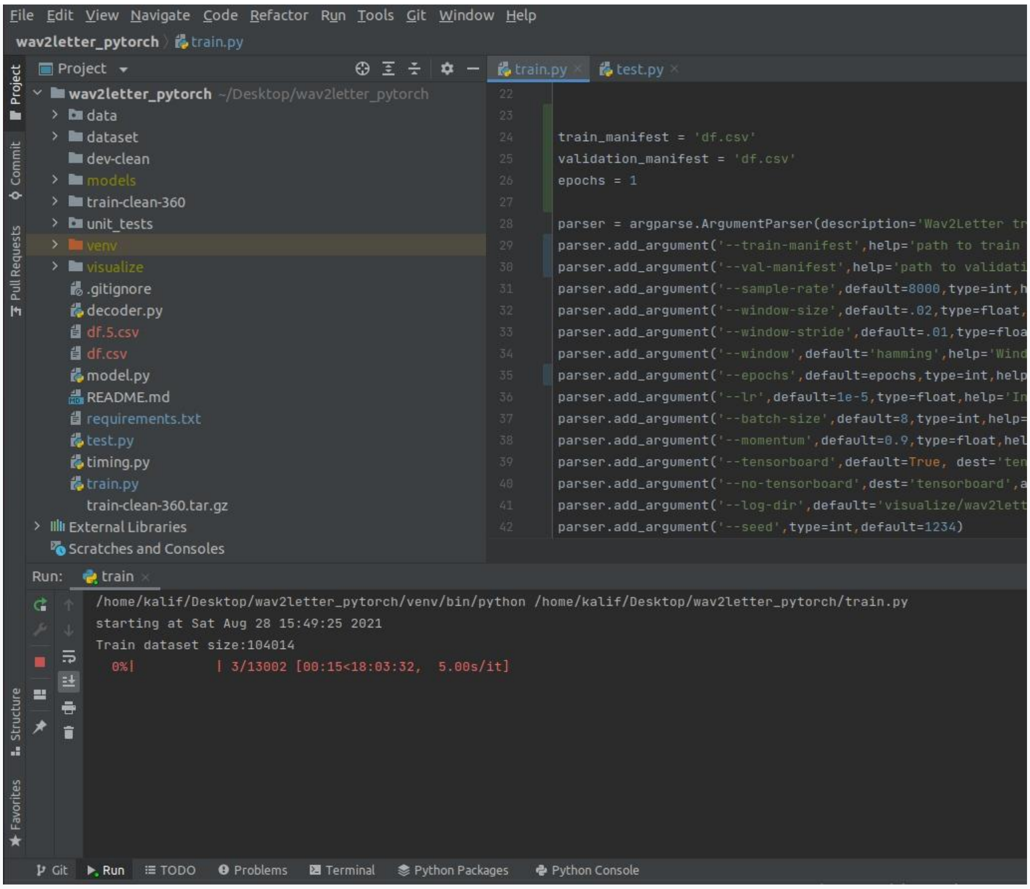
```
epochs = 15
continue_from = True
continue_from_path = 'models/wav2letter/epoch_3.pth'

parser = argparse.ArgumentParser(description='Wav2Letter training')
parser.add_argument('--train-manifest', help='path to train manifest csv', default=train_manifest)
parser.add_argument('--val-manifest', help='path to validation manifest csv', default=validation_manifest)
parser.add_argument('--sample-rate', default=8000, type=int, help='Sample rate')
parser.add_argument('--window-size', default=.02, type=float, help='Window size for spectrogram in seconds')
parser.add_argument('--window-stride', default=.01, type=float, help='Window sstride for spectrogram in seconds')
parser.add_argument('--window', default='hamming', help='Window type for spectrogram generation')
parser.add_argument('--epochs', default=epochs, type=int, help='Number of training epochs')
parser.add_argument('--lr', default=1e-5, type=float, help='Initial learning rate')
parser.add_argument('--batch-size', default=8, type=int, help='Batch size to use during training')
parser.add_argument('--momentum', default=0.9, type=float, help='Momentum')
parser.add_argument('--tensorboard', default=True, dest='tensorboard', action='store_true',
                    help='Turn on tensorboard graphing')
parser.add_argument('--no-tensorboard', dest='tensorboard', action='store_false', help='Turn off tensorboard graphin
parser.add_argument('--log-dir', default='visualize/wav2letter', type=str, help='Directory for tensorboard logs')
parser.add_argument('--seed', type=int, default=1234)
parser.add_argument('--id', default='Wav2letter training', help='Tensorboard id')
parser.add_argument('--model-dir', default='models/wav2letter',
                    help='Directory to save models. Set as empty, or use --no-model-save to disable saving.')
parser.add_argument('--no-model-save', dest='model_dir', action='store_const', const='')
parser.add_argument('--layers', default=1, type=int,
```

After setting the desired parameters, you can run the file

“train.py”

and the training will start.

It’s should seem like that:



3.4 Testing/Inference

To evaluate a trained model on a test set, the set must be in the same format as the training set.

Note to adjust the following parameters in “test.py” file:

1. test_dataset - should be the name of the test file.

2. model_path - the path in the project to the model we want to test.
3. beamSearch – search strategy, considers multiple best options based on beamwidth using conditional probability, which is better than the sub-optimal Greedy search. comma separated value for k,alpha,beta,prune. See example below.

```
test_dataset = 'mycsvfile.csv'
model_path = 'models/wav2letter/epoch_3.pth'
beamSearch = '5,0.3,5,1e-3'
decoderVar = 'greedy'
lmPath = ''
print_samples = False
print_all = True
```

After setting the desired parameters, you can run the file

“test.py”

and the test will start.

It’s should seem like that:

When ‘Decoder result’ is the trained model result.

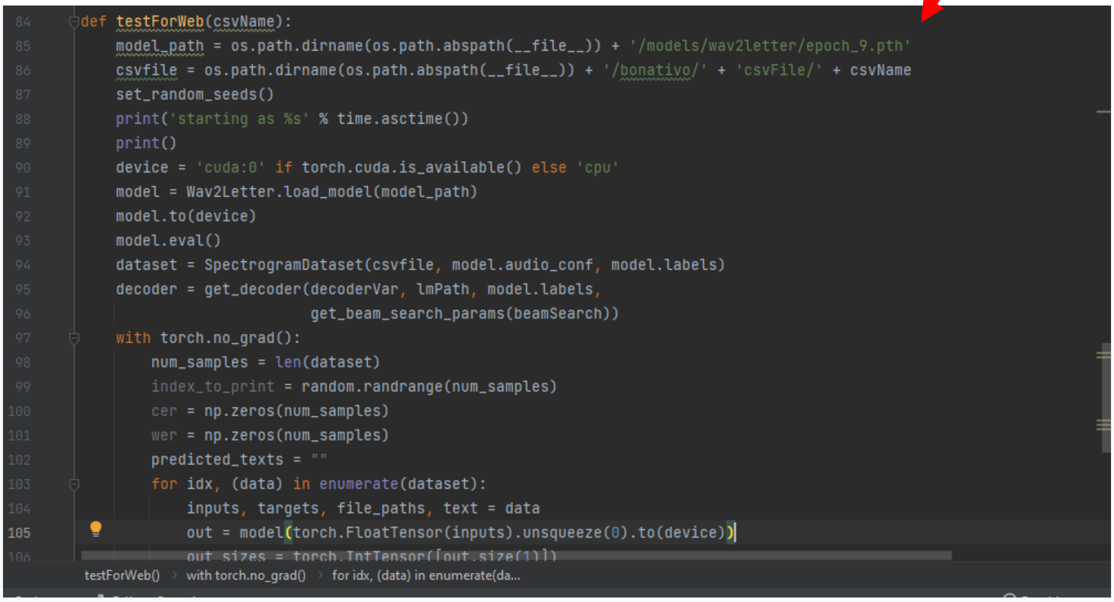
Now the model is ready, and you can use the user interface to see a text translation of audio files.

4. Usage

4.1 User Interface

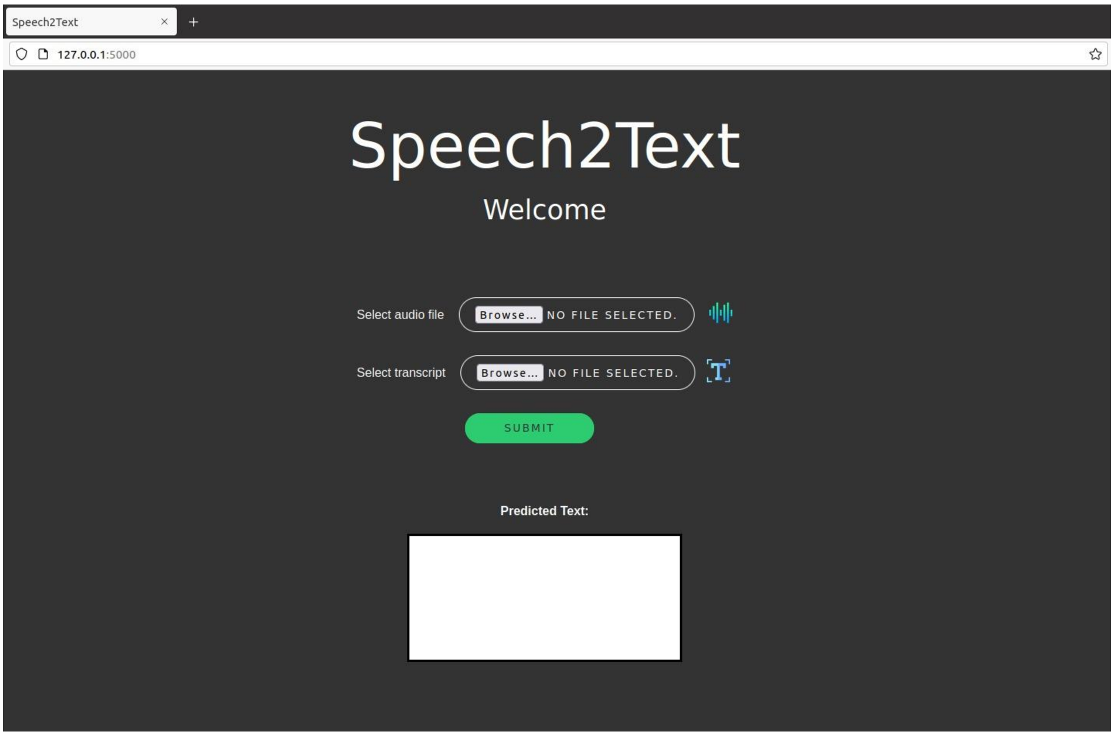
Before opening the interface, enter the file ‘test.py’ file

Define the path of the model you want to use here:



Then, run the file ‘openWeb.py’ and enter the URL you received.

You should see the following page:



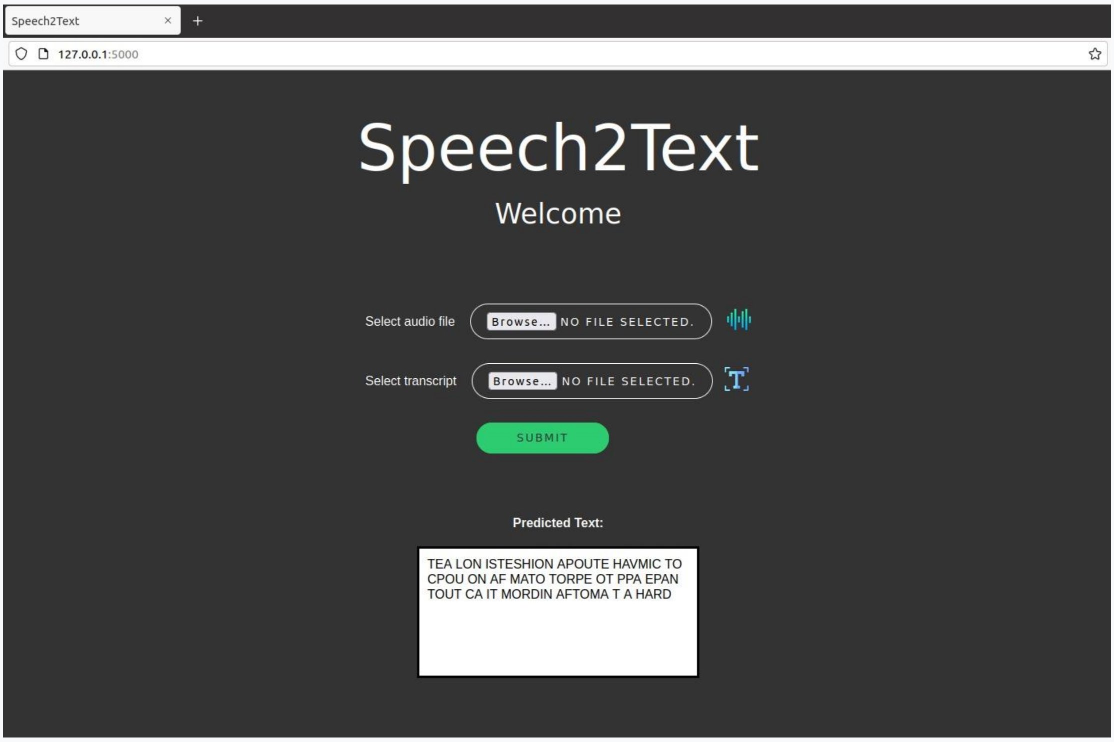
The user interface is simple and easy to use;

An audio file needs to be uploaded with the top button (Note that the file must be .wav or .mp3 file).

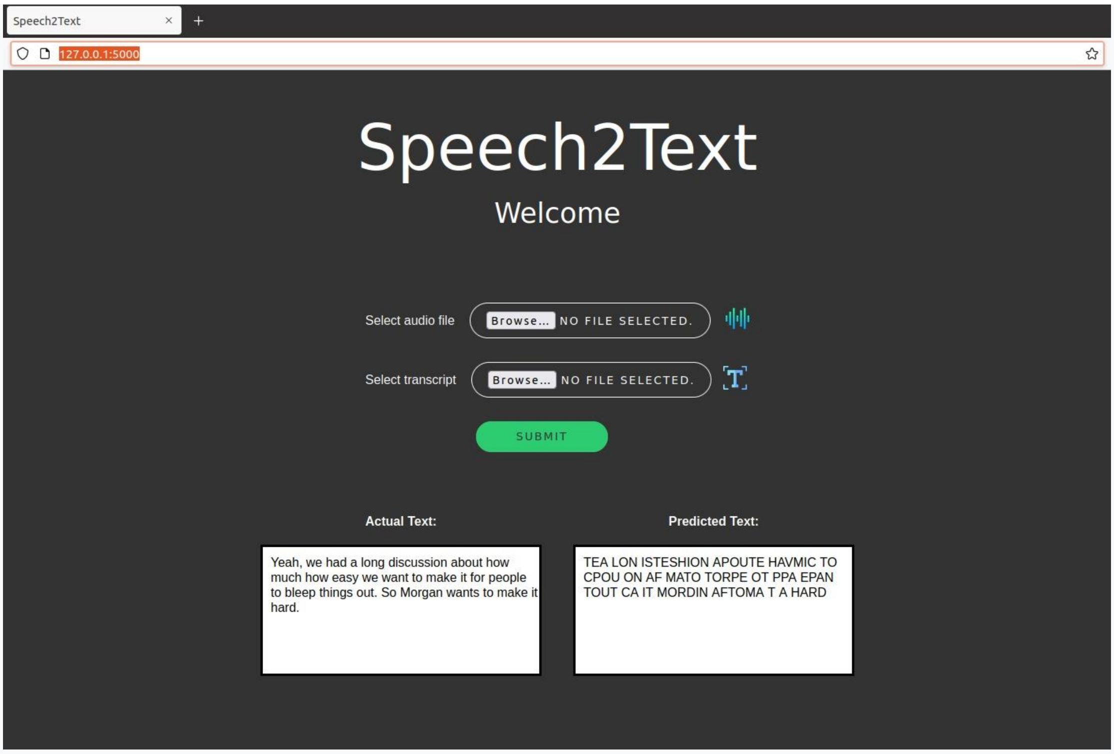
There is an optional button to upload the original text file of the translate of the audio file to compare the predict result to it.

After selecting the appropriate files, press the ’Submit’ button.

The predicted result should be appearing at the white box in the bottom. For example:



If you uploaded the original text too, you may see two white boxes; one for the original translate of the audio and one of the predict. For example:



5. How It Works

5.1 Methods in use

Greedy search decoder -A simple approximation is to use a greedy search that selects the most likely word at each step in the output sequence.

This approach has the benefit that it is very fast, but the quality of the final output sequences may be far from optimal.

CTC lost function - Traditional speech recognition models would require you to align the transcript text to the audio before training, and the model would be trained to predict specific labels at specific frames.

The innovation of the CTC loss function is that it allows us to skip this step. Our model will learn to align the transcript itself during training. The key to this is the “blank” label introduced by CTC, which gives the model the ability to say that a certain audio frame did not produce a character.

SGD algorithm - iterative method for optimizing an objective function with suitable smoothness properties. The use of SGD in the neural network setting is motivated by the high cost of running back propagation over the full training set. SGD can overcome this cost and still lead to fast convergence.

6. Summary

In the project we used existing architectures to build a model that recognizes audio and returns its text translation with Python's pytorch library. You can create and train a model in the project with independent dataset or with LibriSpeech.
In the test/inference stage and in the user interface it is possible to use any trained model that have been trained by this architecture.