Automatic Speech Recognition

Instructor: Dr. Amir Jafari

-Anwesha Tomar

Table of Contents

Abstract:....3
Introduction:....3
Automatic Speech Recognition:....4
Google Speech to Text API:....5
Mozilla DeepSpeech2:....7
Literature Survey:....9
Procedure:....9
Dataset:....9
Ground Truth:....10
DeepSpeech2:....12
Fine Tuning the model:....15
Future Scope:....18
References:....18

Abstract:

In this project automatic speech recognition is implemented using Google Speech to Text API and Mozilla Deepspeech in order to transcribe text from speech. The LM arctic data set nonnative English speech is used for training and testing the model. This dataset consists of speakers who have Arabic, Madrid, Vietnamese, Hindi, Spanish and Korean as their first language. The dataset is preprocessed and then input into the pre-trained model of Deepspeech and Google Speech to Text API.

In this report I describe the pre processing of the data, pre trained models used, the theory behind the models and lastly the future scope.

Introduction:

In this section we describe Automatic Speech Recognition DeepSpeech Model and Google API architecture:

Automatic Speech Recognition:

Speech Recognition is a subfield of computational linguistics that is concerned with recognition and translation of spoken language into text by computers, sometimes referring to the process as "speech to text". Automatic speech recognition (ASR) is a technology that converts the spoken words into text, it requires knowledge about phonetics and phonology; how words are pronounced in terms of sequences of sounds, and how each of these sounds is realized acoustically.

Fig 1: Flow diagram for text to speech

The ASR system has four main components:

- Signal processing
- Feature extraction
- Acoustic model (AM)
- Language model (LM)
- Hypothesis search
-

The signal processing and feature extraction component takes as input the audio signal, enhances the speech by removing noises and channel distortions, converts the signal from time-domain to frequency-domain, and extracts salient feature vectors that are suitable for the following acoustic models.

The acoustic model integrates knowledge about acoustics and phonetics, takes as input the features generated from the feature extraction component, and generates an AM score for the variable-length feature sequence. The language model estimates the probability of a

hypothesized word sequence, or LM score, by learning the correlation between words from a (typically text) training corpora.

The LM score often can be estimated more accurately if the prior knowledge about the domain or task is known. The hypothesis search component combines AM and LM scores given the feature vector sequence and the hypothesized word sequence, and outputs the word sequence with the highest score as the recognition result. In this book, we focus on the AM component.

The variability in the audio signals is caused by complicated interaction of speaker characteristics (e.g., gender, illness, or stress), speech style and rate, environment noise, side talks, channel distortion (e.g., microphone difference), dialect differences, and nonnative accents. A successful speech recognition system must contend with all of these acoustic variabilities.

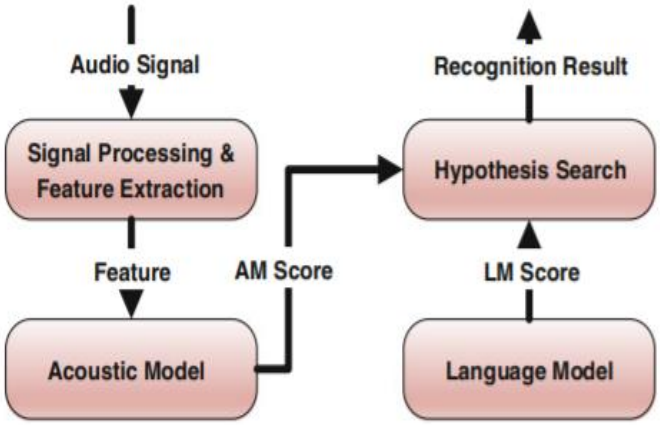

Fig 2: Automatic Speech Recognition pipeline

Google Speech to Text API:

The Asynchronous speech recognition system sends audio data to the Speech-to-Text API and initiates a Long Running Operation. Using this operation, we transcribe the audio files, it can process up to 1 minute of speech audio data.

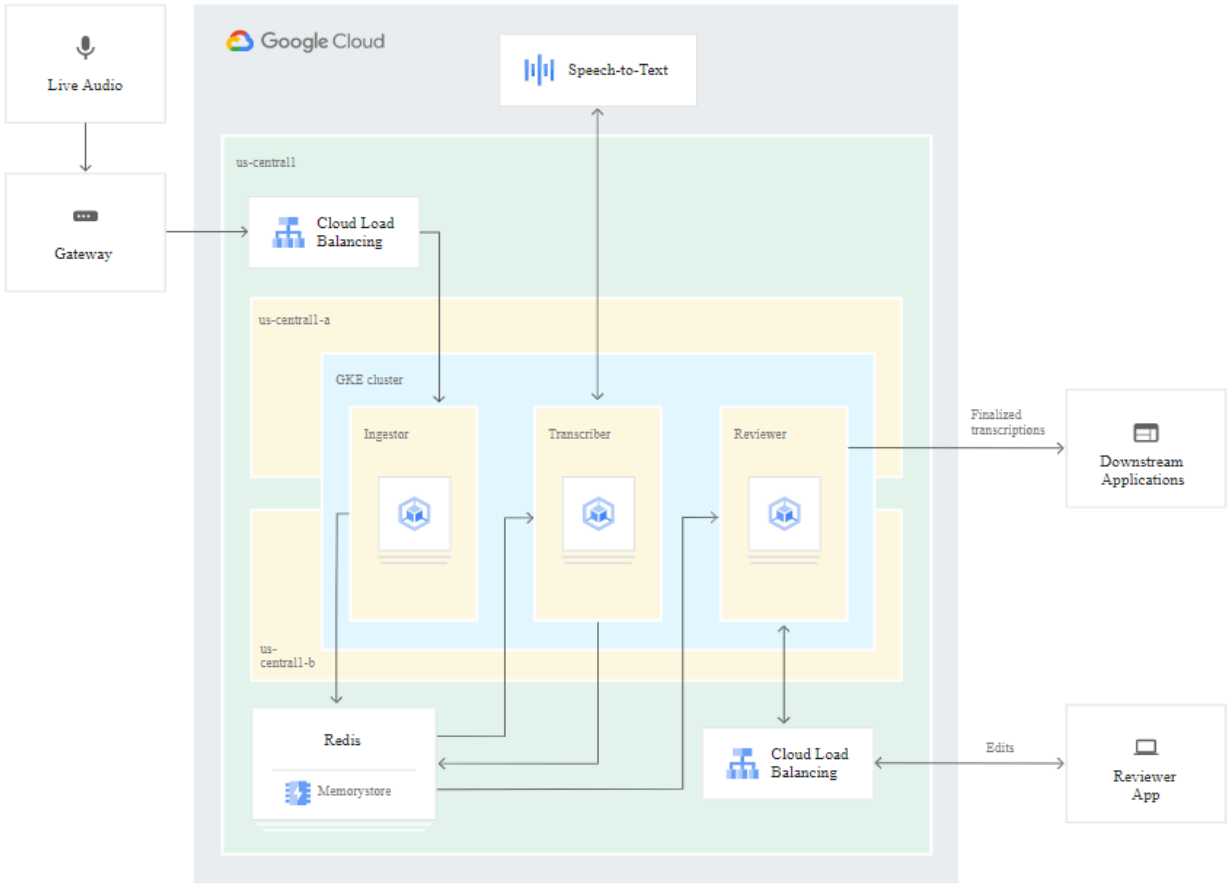

Fig 2. Google Speech to Text Archtiechture

Description of the components of the Google Speech to Text API:

Ingestor	Consumes the source audio stream and writes to intermediate storage. The audio might be delivered using third-party software and domain-specific protocols, and might also be encrypted. Therefore, it makes sense to design and deploy the ingestion component separately from the transcription component.

Transcriber	Consumes the ingested audio, calls Speech-to-Text, and writes transcription results to intermediate storage. The Transcriber service is the core of the app. It's responsible for managing and maintaining connections to Speech-to-Text and for processing transcription results.
Reviewer	Consumes the transcriptions and displays them in a web app for review. Typically, media organizations require a human operator to oversee the generated transcriptions. To help with this requirement, the Reviewer web app might provide the ability to amend transcriptions before they are sent downstream for broadcast.

Mozilla DeepSpeech2:

The Mozilla Deepspeech2 is an RNN model composed of 5 layers of hidden units, it makes use of Adam optimizer and Relu as an activation function.

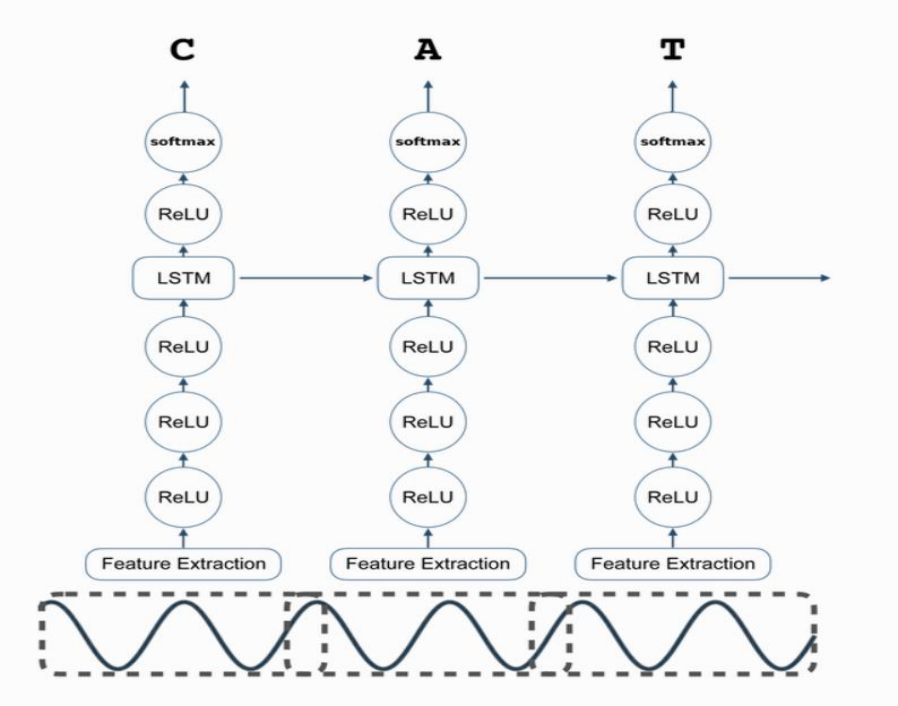

Fig 3. Deepseepch2 Architecture

Deepspeech makes use of a huge amount of data for training, a dropout rate between 5% - 10% is applied to the feed forward layers but not to the recurrent hidden activations. For Deepspeech's English model an extensive dataset consisting of 5000 hours of read speech from 9600 speakers.

Performance metrics for Deepspeech:

System	Clean (94)	Noisy (82)	Combined (176)
Apple Dictation	14.24	43.76	26.73
Bing Speech	11.73	36.12	22.05
Google API	6.64	30.47	16.72
wit.ai	7.94	35.06	19.41
Deep Speech	**6.56**	**19.06**	**11.85**

Deepspeech has also been trained on accented speech, the publicly available VoxForge (http://www.voxforge.org) dataset was used to train. The accents are split into four categories, a test set is created from the VoxForge data with 1024 examples from each accent group forming a total of 4096 examples.

Performance of Deepspeech on accented speech:

Accented Speech			
Test set	DS1	DS2	Human
VoxForge American-Canadian	15.01	7.55	4.85
VoxForge Commonwealth	28.46	13.56	8.15
VoxForge European	31.20	17.55	12.76
VoxForge Indian	45.35	22.44	22.15

We can see that human transcription has a lower WER in comparison to Deepspeech 2. The research we attempted to conduct was to improve Deep Speech 2 on non-native English speaker dataset.

This research project was inspired by Deepspeech inability to correctly transcribe accented data.

Literature Survey:

The very first model that I studied used Recurrent Neural Networks to evaluate and transcribe the speakers native language and the English language. The data used was Spanish languages and Indian languages along with English. The best performance by this model yields a 11.95 Word Error Rate for Hispanic accents and 17.55 Word Error Rate for Indian accents. The adjusted model that was defined later yields a lower Word Error Rate of 12 and 8.4 for Hispanic and Indian accents each.

The next model used real time transcriptions, the goal of this paper was to analyze three conditions. The first condition was when a perfect transcript already exists, the next was with a imperfect transcript, the last condition was when no transcript exists. It was noted that the average Word Error Rate was 20 and the lowest Word Error Rate was 10. The conclusion was that a model can only be trained well when a perfect target transcript exists. In the lack of a perfect transcript the model fails to perform well.

The last paper explore the German language, a model was designed to test bilingual speech data. The final Word Error Rate was 42.7 after pooling different models a toolkit was designed called James Recognition Toolkit.

After exploring these models we moved forward with exploring different non native speaker datasets to test pre trained models and continue with fine tuning models.

Procedure:

The complete code and instructions to execute it can be found here.

Dataset:

The dataset was collected from TAMU Arctic Corpus, this dataset consists of a collection of non native English speakers, the data was collected from Arabic, Mandarin, Hindi, Korean, Spanish and Vietnamese speakers. It consists of an equal number of males and females. The speech was recorded at 44kHz and phonetically-balanced short sentences, the duration of each speech recording is under 6 seconds.

The speech was collected from 25 people and each file consists of four sub files namely, annotation, textgrid, transcripts and wav files. In the research transcripts and wav files were used. The transcription files were pre processed by converting the text to lower case. The wav files were pre processed from 16kHz to 44KHz.

Ground Truth:

The Google Speech API was used to find the ground truth. The following code was used to input the wav files into the API and then produce transcripts.

```
def transcribe_file(speech_file):
    """Transcribe the given audio file."""

    client = speech.SpeechClient()

    with io.open(speech_file, "rb") as audio_file:
        content = audio_file.read()

    audio = speech.RecognitionAudio(content=content)
    config = speech.RecognitionConfig(
        encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,
        sample_rate_hertz=44100,
        language_code="en-US",
    )
```

Next we store the produced transcripts in a dataframe, for comparison later.

```
    DATA_DIR = os.getcwd()+ "/wav/"
    for f in os.listdir(DATA_DIR):
        if f[-4:] == ".wav":
            g_name.append(f)
            g_trans.append(transcribe_file(DATA_DIR+f))


    df['transcript'] = g_trans
    df['wav'] = g_name
    df.to_pickle("/home/anwesha/speechrecog/google_api/gapi.pkl")
    print(df)
```

Lastly we produce the Word Error Rate.

```
for f in os.listdir(DATA_DIR):
        for i in range(len(df)):
            temp=df['wav'][i]
            if (f[:-4]== temp[:-4]):
                error_trans.append(temp)
                error.append(wer(DATA_DIR + f, df['transcript'][i]))
    df_error['WAV']=error_trans
    df_error['WER']=error
```

```
df_error.to_pickle("/home/anwesha/speechrecog/google_api/gapi_WER.pkl")
```

The two dataframes, one for storing transcripts and the other for storing the word error rate are displayed below.

Transcripts:

	transcript	wav
0	how could explain his position over the sketch	arctic_a0042.wav
1	in a Flash clip followed its direction	arctic_a0068.wav
2	his Immaculate appearance was gone	arctic_a0059.wav
3	this has come with terrible sadness	arctic_a0086.wav
4	perhaps she had already met her faith a little...	arctic_a0096.wav
..	...	...
95	is right over there obscurity the words affect...	arctic_a0050.wav
96	you went to that one in Midstream searching th...	arctic_a0092.wav
97	from that moment his friendship for Plies turn...	arctic_a0017.wav
98	yes it was a man who asked a stranger	arctic_a0063.wav
99	he looking at the handkerchief more closely	arctic_a0048.wav

[100 rows x 2 columns]

WER:

	WAV	WER
0	arctic_a0025.wav	10.0
1	arctic_a0073.wav	12.0
2	arctic_a0086.wav	6.0

3 arctic_a0046.wav 11.0

4 arctic_a0021.wav 7.0

.. ..

95 arctic_a0001.wav 8.0

96 arctic_a0060.wav 9.0

97 arctic_a0018.wav 8.0

98 arctic_a0037.wav 8.0

99 arctic_a0022.wav 13.0

[100 rows x 2 columns]

Final Error Rate: 9.06

DeepSpeech2:

Next we made use of Mozilla Deep Speech's pretrained(0.9.1) model, in order to generate transcripts. The following code, is used transcribe text from speech.

```
os.chdir("/home/anwesha/")
for i in range(1,val):
    if (i>=1 and i<=9):
        temp=subprocess.check_output("deepspeech --model deepspeech-0.9.1-models.pbmm "
                                     "--scorer deepspeech-0.9.1-models.scorer "
                                   "--audio /home/anwesha/Automatic-Speech-
Recognition/speechrecog/deepspeech/wav/arctic_a000{}.wav".format(i),shell=True)
        temp=str(temp)
        n_line='\n'
        n_line=str(n_line)
        temp=temp.lstrip("b").strip("'").strip(n_line)
        g_trans.append(temp)
        g_name.append("arctic_a000{}.wav".format(i))
    elif (i>=10 and i<=99):
         temp = subprocess.check_output("deepspeech --model deepspeech-0.9.1-models.pbmm "
                                    "--scorer deepspeech-0.9.1-models.scorer "
                                    "--audio /home/anwesha/Automatic-Speech-
Recognition/speechrecog/deepspeech/wav/arctic_a00{}.wav".format(i),
```

```
                                shell=True)
        temp = str(temp)
        n_line = '\n'
        n_line = str(n_line)
        temp = temp.lstrip("b").strip("'").strip(n_line)
        g_trans.append(temp)
        g_name.append("arctic_a00{}.wav".format(i))
    elif (i>=100 and i<=999):
        temp = subprocess.check_output("deepspeech --model deepspeech-0.9.1-models.pbmm "
                                "--scorer deepspeech-0.9.1-models.scorer "
                                "--audio /home/anwesha/Automatic-Speech-
Recognition/speechrecog/deepspeech/wav/arctic_a0{}.wav".format(i),
                                shell=True)
        temp = str(temp)
        n_line = '\n'
        n_line = str(n_line)
        temp = temp.lstrip("b").strip("'").strip(n_line)
        g_trans.append(temp)
        g_name.append("arctic_a0{}.wav".format(i))
```

These transcripts are then stored in a dataframe.

```
df['wav']=g_name
df['transcript']=g_trans
df.to_pickle("/home/anwesha/speechrecog/deepspeech/dpspeech.pkl")
print(df)
```

To find the Word Error Rate we compare the generated transcripts with the following code is used:

```
for f in os.listdir(DATA_DIR):
    for i in range (len(df)):
        temp = df['wav'][i]
        if (f[:-4]== temp[:-4]):
            error_trans.append(temp)
            error.append(wer(DATA_DIR + f, df['transcript'][i]))
df_error['Wav']=error_trans
df_error['WER']=error
```

The two dataframes, one for storing transcripts and the other for storing the word error rate are displayed below.

Transcripts:

	wav	transcript
0	arctic_a0001.wav	author of the lancastrian flit steals extra\n

1 arctic_a0002.wav not at this particular case tom a rigid with m...

2 arctic_a0003.wav that went it time that evening the two men sho...

3 arctic_a0004.wav "lord but i'm glad to see you again fell\n"

4 arctic_a0005.wav we ever forget it\n

.. ..

94 arctic_a0095.wav a big canvas tent toward the first thing to co...

95 arctic_a0096.wav but has she had already met her fate a little ...

96 arctic_a0097.wav then you can arrange yourself comfortably amon...

97 arctic_a0098.wav shut i carry you\n

98 arctic_a0099.wav mid the ink joy pounded in his brain\n

[99 rows x 2 columns]

WER:

Wav WER

0 arctic_a0025.wav 10.0

1 arctic_a0073.wav 12.0

2 arctic_a0086.wav 5.0

3 arctic_a0046.wav 8.0

4 arctic_a0021.wav 7.0

.. ..

94 arctic_a0001.wav 7.0

95 arctic_a0060.wav 9.0

96 arctic_a0018.wav 6.0

97 arctic_a0037.wav 8.0

98 arctic_a0022.wav 12.0

[99 rows x 2 columns]

Final Error Rate: 8.737373737373737.

The Deepspeech model performs better, therefore we continue with improving and fine tuning this model.

Fine Tuning the model:

In order to improve the model for the non native English speakers dataset, we need to create a two files, an alphabet file and a train file.

For fine tuning Deepspeech a new language model is to be created; this language model requires an alphabet input. The alphabets are extracted from the given target transcripts. The following code is used to find the alphabets from the given transcripts.

```
import argparse
import csv
import os
import sys
import unicodedata
from io import open

def main():
    parser = argparse.ArgumentParser()

    parser.add_argument("-csv", "--csv-files", help="Str. Filenames as a comma separated list",
required=True)
    parser.add_argument("-alpha", "--alphabet-format", help="Bool. Print in format for alphabet.txt",
action="store_true")
    parser.add_argument("-unicode", "--disable-unicode-variants", help="Bool. DISABLE check for unicode
consistency (use with --alphabet-format)", action="store_true")
    args = parser.parse_args()
    in_files = args.csv_files.split(",")

    print("### Reading in the following transcript files: ###")
    print("### {} ###".format(in_files))

    all_text = set()
    for in_file in in_files:
        with open(in_file, "r") as csv_file:
            reader = csv.reader(csv_file)
            try:
                next(reader, None)  # skip the file header (i.e. "transcript")
                for row in reader:
                    if not args.disable_unicode_variants:
                        unicode_transcript = unicodedata.normalize("NFKC", row[2])
```

```
                    if row[2] != unicode_transcript:
                        print("Your input file", in_file, "contains at least one transript with unicode chars on more
than one code-point: '{}'. Consider using NFKC normalization: unicodedata.normalize('NFKC',
str).".format(row[2]))
                        sys.exit(-1)
                    all_text |= set(row[2])
            except IndexError:
                print("Your input file", in_file, "is not formatted properly. Check if there are 3 columns with the
3rd containing the transcript")
                sys.exit(-1)
            finally:
                csv_file.close()

    print("### The following unique characters were found in your transcripts: ###")
    if args.alphabet_format:
        for char in list(all_text):
            print(char)
            print("\n")
        print("### ^^^ You can copy-paste these into data/alphabet.txt ###")
    else:
        for l in all_text:
            print(l)

if __name__ == '__main__':
    main()
```

Next we produce the train file, using the following code. The train file consists of three columns wav_filename,wav_filesize, transcript. The wav_filename column consists of the path to each wav file, while the wav_filesize, displays the size of each wav file. The last column transcript is used by the model for fine tuning purposes.

```
import re
import os
import pandas as pd
import csv
import unicodedata
import warnings
warnings.simplefilter("ignore")

#creating data frame with target transcripts
DATA_TEXT=os.getcwd()+"/transcript/"
trans=[]
txt_name=[]
for path in os.listdir(DATA_TEXT):
    with open(DATA_TEXT + path[:-4] + ".txt", "r") as s:
        trans.append(s.read())
        txt_name.append(path)

df_target=pd.DataFrame()
df_target['file name']=txt_name
df_target['transcript']=trans
```

```
df_target.to_pickle("/home/anwesha/Automatic-Speech-
Recognition/speechrecog/deepspeech/target_transcripts.pk")

data=pd.read_pickle("dpspeech.pkl")
rows=len(data)
file_stats=[]
DATA_DIR = os.getcwd()+ "/wav/"
for f in os.listdir(DATA_DIR):
    file_stats.append(os.stat(DATA_DIR+f).st_size)
data['file_size']=file_stats[:99]
FIELDNAMES = ["wav_filename", "wav_filesize", "transcript"]
target_csv="train.csv"
with open(target_csv, "w", encoding="utf-8", newline="") as target_csv_file:
    writer = csv.DictWriter(target_csv_file, fieldnames=FIELDNAMES)
    writer.writeheader()
    for i in range(0,len(data['wav'])):
        for j in range(0,len(df_target['file name'])):
            t1=data['wav'][i]
            t2=df_target['file name'][j]
            if (t1[:-4] == t2[:-4]):
                data['transcript'][i]=(df_target['transcript'][j].lower())
    #bar = progressbar.ProgressBar(max_value=len(rows), widgets=SIMPLE_BAR)
    for w1,v,t in zip(data['wav'],data['file_size'],data['transcript']):
        writer.writerow(
                {
                    "wav_filename": DATA_DIR+w1,
                    "wav_filesize":v,
                    "transcript": t,
                }
            )
```

Lastly we fine tune the model using the following code.

```
import subprocess
subprocess.check_output("python3  /home/anwesha/DeepSpeech/DeepSpeech.py "
                        "--audio_sample_rate 16000 "
                        "--alphabet_config_path /home/anwesha/Automatic-Speech-
Recognition/speechrecog/deepspeech/alphabet.txt "
                        "--save_checkpoint_dir /home/anwesha/Automatic-Speech-
Recognition/speechrecog/deepspeech/checkpoint "
                        "--dropout_rate 0.05 "
                        "--train_files /home/anwesha/Automatic-Speech-
Recognition/speechrecog/deepspeech/train.csv "
                        "--export_dir /home/anwesha/Automatic-Speech-
Recognition/speechrecog/deepspeech/models",shell=True)
```

This part of the research I have not yet been able to work out.

Future Scope:

I would first attempt to make Deepspeech fine tuning work in tensorflow. Next I would explore different frameworks such as Tensorflow. Lastly, I would collect more data in order to improve the model.

References:

1. Leveraging native language information for improved accented speech recognition. (2019). *ArXiv*, 1. https://arxiv.org/abs/1904.09038

2. Effects of Automated Transcription Quality on Non-native Speakers' Comprehension in Real-time Computer-mediated Communication. (2010). *CHI 2010: Sound and Speech*, 1725. https://www.researchgate.net/publication/221514901_Effects_of_automated_transcription_quality_on_non-native_speakers'_comprehension_in_real-time_computer-mediated_communication

3. Wang,Schultz,Waibel, Z. T. A. (2003). COMPARISON OF ACOUSTIC MODEL ADAPTATION TECHNIQUES ON NON-NATIVE SPEECH. *ICASSP*, 540. https://www.researchgate.net/publication/4015124_Comparison_of_acoustic_model_adaptation_techniques_on_non-native_speech