Speech to Text conversion of NPTEL Lectures using ASR-ESPNET

A Project Report

submitted by

POSINA SHANMUKA BHARATH KUMAR

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **Speech to Text conversion of NPTEL Lectures using ASR-ESPNET**, submitted by **Posina Shanmuka Bharath Kumar**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Umesh.S
Research Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 10th May, 2019

ACKNOWLEDGEMENTS

I take this opportunity to express my deepest gratitude to my project guide Prof. Umesh.S for his valuable guidance and motivation throughout the project. I am very grateful to him for providing his valuable time to guide me during the project. It is a privilege to be a student in IIT Madras. I express special thanks to all my teachers for all the academic insight obtained from them. I also acknowledge the excellent facilities provided by the institute to the students.

I am indebted to my parents and my brother for their unconditional love, support and guidance.

ABSTRACT

KEYWORDS: ASR-ESPNET

The National Programme on Technology Enhanced Learning (NPTEL) is an initiative in which several Indian Institutes of Technology (IIT Bombay, Delhi, Guwahati, Kanpur, Kharagpur, Madras and Roorkee) and the Indian Institute of Science (IISc, in Bangalore) are partners in creating complete, free and open course ware online for engineering, science and management subjects, and in training teachers in Indian technical institutions to help improve the overall quality of technical and professional education and the employ-ability of Indian graduates. The contents are, however, available free to everyone in the world and follow closely the curriculum design adopted by major technical universities in India and abroad.

Most of the lectures that are available on NPTEL are in English language. There are many students who struggle with English language while attending NPTEL lectures online. They might benefit if these lectures are translated into different languages. Currently We have Sophisticated Language models that we are capable of doing speech recognition very well and all that we need is Data to train them.

My project is a part of above main idea that helps to convert Audio to text of the lectures by using ASR-ESPNET. I trained and decoded the audio files using ESPNET architecture on a large dataset form the NPTEL websites and train them by using the time frame information available for each lecture in the form of “srt” files(subtitles file).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

ASR	Automatic Speech Recognition
WER	Word Error Rate
CER	Character Error Rate
BLSTM	Bidirectional Long Short-Term Memory
RNN	Recurrent Neural Network

NOTATION

.srt	Subtitle file extension
.mp3	Audio file extension

CHAPTER 1

INTRODUCTION

1.1 ESPNET

ESPnet is an end-to-end speech processing toolkit, mainly focuses on end-to-end speech recognition and end-to-end text-to-speech. ESPnet uses chainer and pytorch as a main deep learning engine, and also follows Kaldi style data processing, feature extraction/-format, and recipes to provide a complete setup for speech recognition and other speech processing experiments.

Automatic speech recognition (ASR) becomes a mature technology with a lot of research and development efforts mainly in speech processing communities. This paper describes a new open source toolkit named ESPnet (End-to-end speech processing toolkit), which aims to provide a neural end-to-end platform for ASR and other speech processing.ESPnet provides a single neural network architecture to perform speech recognition in an end-to-end manner.

ESPnet fully utilizes benefits of two major end-to-end ASR implementations based on both connectionist temporal classification (CTC) and attention-based encoder-decoder network. Attention-based methods use an attention mechanism to perform alignment between acoustic frames and recognized symbols.

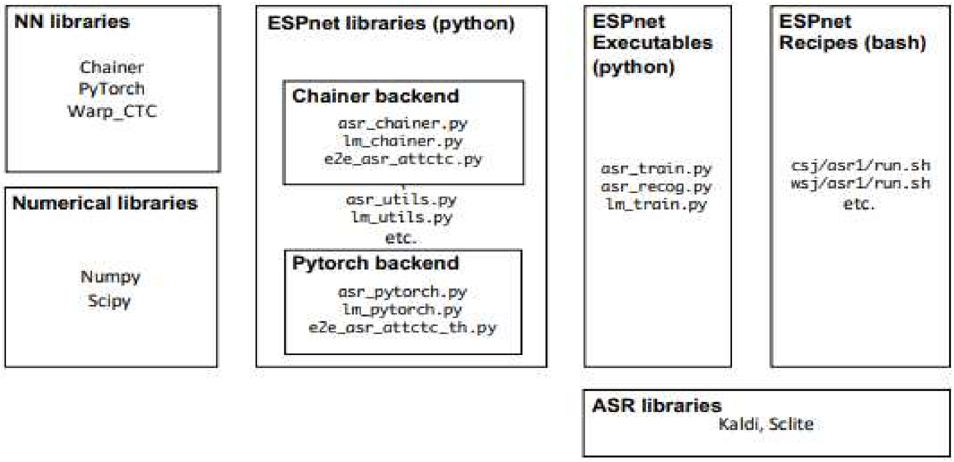

Figure 1.1: Software architecture of ESPnet

1.2 Functionality

Figure 1.1 shows a software architecture of ESPnet. In the ESPnet, main neural network training and recognition parts are written in python, which calls Chainer and PyTorch by switching the backend option.

1.2.1 Attention-based encoder-decoder

Encoder

The default encoder network is represented by bidirectional long short-term memory (BLSTM) with subsampling given T-length speech feature sequence $O_{1:T}$ to extract high-level feature sequence $h_{1:T'}$ as

$$h_{1:T'} = BLSTM(O_{1:T})(1.1)$$

where $T^{'} < T$ in general due to the subsampling.

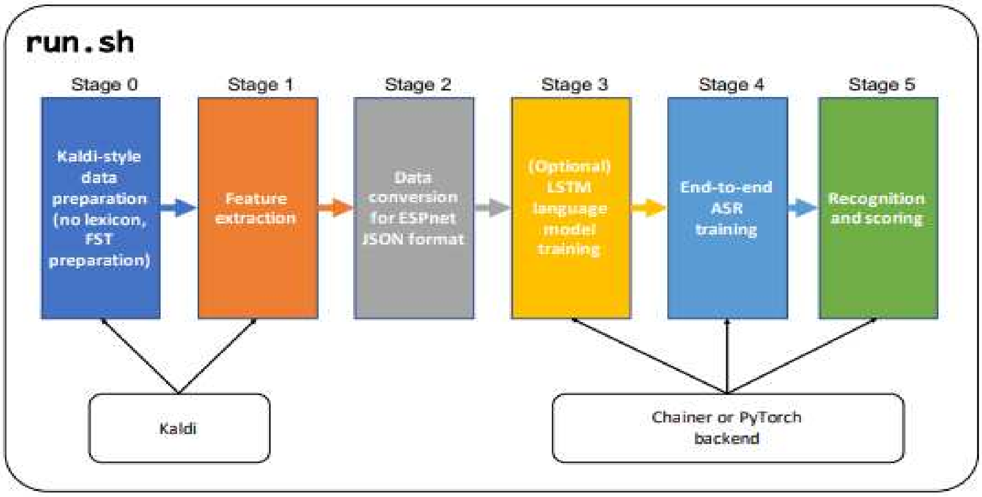

Figure 1.2: Flow of standard ESPnet run.sh code

1.3 run.sh

1.3.1 code

`run.sh` code is a shell script.

1.3.2 Standard flow

Figure 1.2 shows a flow of standard recipes in ESPnet.The standard recipe includes the following 5 stages in `run.sh`:

Stage 0(optional) - Data Download

This stage is for data download if we dont have data. We adopt the Kaldi data directory format, and we can simply use the Kaldi data preparation script.

stage 1: Feature Generation

we use the Kaldi feature extraction. Most of recipes use the 80-dimensional log Mel feature with the pitch feature (totally 83 dimensions). These features are stored in fbank directory.

stage 2: Dictionary and Json Data Preparation

This stage converts all the information including in the Kaldi data directory (transcriptions, speaker and language IDs, and input and output lengths) to one JSON file (`data.json`) except for input features.

stage 3: Network Training

Character-based BLSTM is trained by using either Chainer or PyTorch backend. Attention-based encoder-decoder is trained by using either Chainer or PyTorch backend.

stage 4: Decoding

After training the model, testing of model is done by decoding on seperate data set. After decoding, results are produced in terms of WER(Word Error Rate), CER(Charecter Error Rate).

CHAPTER 2

Data Preprocessing before training

2.1 Introduction

Data required for model training are audio files(.wav format) and corresponding subtitle file(.srt file). These data can be downloaded from NPTEL website. But these audio files are mostly 50min - 1hour long files. For training we need audio files of length approximately 15 - 20sec and corresponding subtitle files. This segmentation is done by split_srt.py python code.

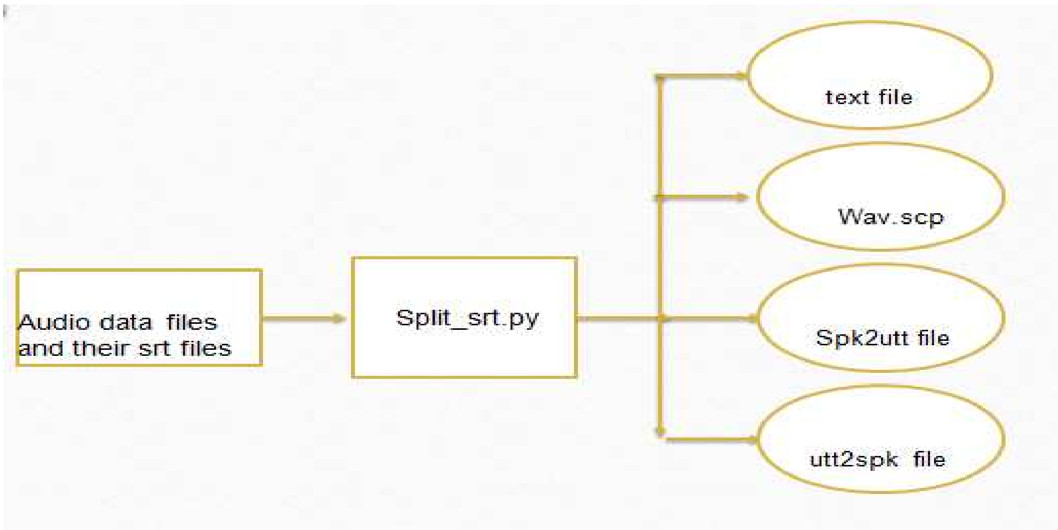

Figure 2.1: Flow chart of inputs and outputs of split-srt.py

2.2 Input formats required for model training

Model training requires certain formats of inputs. The following are required formats:

text.txt

This text file should contain Utterance ID followed by the text of segmented audio file. An example line in text file :

mod01lec01_1006.72_1019.29 in India there are many factors that can influence the society

where,
mod01lec01_1006.72_1019.29 - Utterance ID
mod01lec01 - lecture number,
1006.72 - start time in seconds,
1019.29 - end time in seconds.
At last is the text data in that time segment.

wav.scp

The wav.scp file should contain Utterance ID followed by entire path of corresponding audio file.

An example line in wav.scp file :
mod01lec01_1006.72_1019.29 /speech/batch1/bharath/testnptel5/data/tr/
splitwav/ mod01lec01_1006.72_1019.29. wav

where,
mod01lec01_1006.72_1019.29 - Utterance ID

spk2utt file

The spk2utt file should contain Speaker ID followed by each Utterance ID.

utt2spk file

The utt2spk file should contain each Utterance ID followed by speaker ID.

where

Speaker ID: ID given to each speaker(in this case lecturer)

Utterance ID: ID give to each segmented audio file in the format of professor_name_course_start-time_end-time

2.3 Algorithm of split_str.py

The algorithm of split_srt.py code is as follows:

```
mp3_list = contains all the names of the mp3 files in the current folder.
extract_timeframes_text() : function that returns a list of timestamp information and corresponding text data from the given srt file.
time_framelist=contains the timestamps
text_list =contains the corresponding text data for each timestamp
Algo:
for mp3_file in mp3_list:
initialise time_framelen_tillnow=0
initialise prev_timestamp=0
initialise clip_start=0
time_framelist ,text_list =extract_timeframes_text(mp3_list+ .srt )
text=
for i in range(len(time_framelist)):
temp_timeframe_len=time_framelist[i]-prev_timestamp
if(temp_timeframe_len+time_framelen_tillnow <=15):
time_framelen_tillnow+=temp_timeframe_len
text+=text_list[i]
prev_timestamp=time_framelist[i]
else:
File.write(textfile , clip_start , prev_timestamp, text_list)
File.write(spk2_utt , clip_start , prev_timestamp)
File.write(utt2spk , clip_start , prev_timestamp)
```

```
File.write(wav.scp , clipped paths of wav files, clip_start ,
    prev_timestamp)
clip_start=prev_timestamp
time_framelen_tillnow=0
temp_timeframe_len=time_framelist[i]-prev_timestamp
time_framelen_tillnow+=temp_timeframe_len
prev_timestamp=time_framelist[i]
```

Note:

- Several corner cases and how the text , wav.scp ,spk2utt, utt2spk files were written were not addressed here.
- The clipped audio segments cannot directly be given as input to the espnet because they are generally sampled at different rates (40 to 45 kHz) which unnecessarily uses a lot of space so all the clipped audio segments are downsampled to 16 KHz using sox command in resample.sh code.
- All the clipped .wav files are indexed to a single speaker itself in spk2utt ,utt2spk files because we don't have the information about the course instructors for a course in the downloaded data. If so , then index each course by a different speaker.

2.4 Key procedures that are to be taken care of before running split_srt.py

There are some processes that are to be taken care of before running split_srt.py. They are as follows:

- Download mp3 files and the corresponding srt files for each lecture from NPTEL website.
- Preprocess the srt files to check for unusual tokens because they reduce the model performance.
- Naming for both mp3 and srt files of a lecture that is downloaded from NPTEL website should be same. This is the important one since it is assumed in the code.

After running split_srt.py, we get all the input files required for ESPnet model training. Train the ESPnet model using the input files.

CHAPTER 3

Model Performance after Training

After training the model, model is tested by decoding it with some data files. The results of decoding are as follows:

3.1 The model performance when trained for 8 hours data and tested for 1hour data using RNN model

3.1.1 Without text normalisation

Table 3.1: The model performance without text normalisation

Word error rate	Character error rate	Number of epochs
89.2	56.5	15

3.1.2 With text normalisation

Table 3.2: The model performance with text normalisation

Word error rate	Character error rate	Number of epochs
44.1	20.5	15

As we can see from Table3.1 and Table 3.2 that there is significant decrease in Word Error rate and Character Error Rate from training without normalisation to with normalisation. This is because in training without normalisation there will be capital letters and small letters and words using them. since there is less data and more targets hence WER and CER is high. Where as in the case of with normalisation, there will be only either Upper case or lower case words. Hence, WER and CER decreased greatly.

3.1.3 Observations

The word error rates have been same when we normalised the data into all capital letters to the case where data is normalised to all small letters.

3.2 The model performance when trained for 8 hrs data and tested for one hour data using BLSTM

Table 3.3: The model performance when trained for 8 hrs data and tested for one hour data using BLSTM

WER for validation	CER for validation	CER for training	Number of epochs
44.1	20.5	8.6	15

As we can see from Figure 3.1, the loss decreases with number of epochs. This is because since there is not lots of data, it requires lots of epochs. This effect can also be seen in Figure 3.2- Accuracy increases with number of epochs and from Figure 3.3- CER decreases with number of epochs.

3.3 The model performance when trained for 400 hrs data and tested for one hour data using BLSTM

Table 3.4: The model performance when trained for 400 hrs data and tested for one hour data using BLSTM

WER for validation	CER for validation	CER for training	Number of epochs
19.6	9.8	4.8	11

In this case, there is lots of data, so training doesn't require many number of epochs. This can be seen in Figure 3.4 loss increases with epochs, from Figure 3.5 accuracy decreases with number of epochs and from Figure 3.6 CER increases with number of epochs.

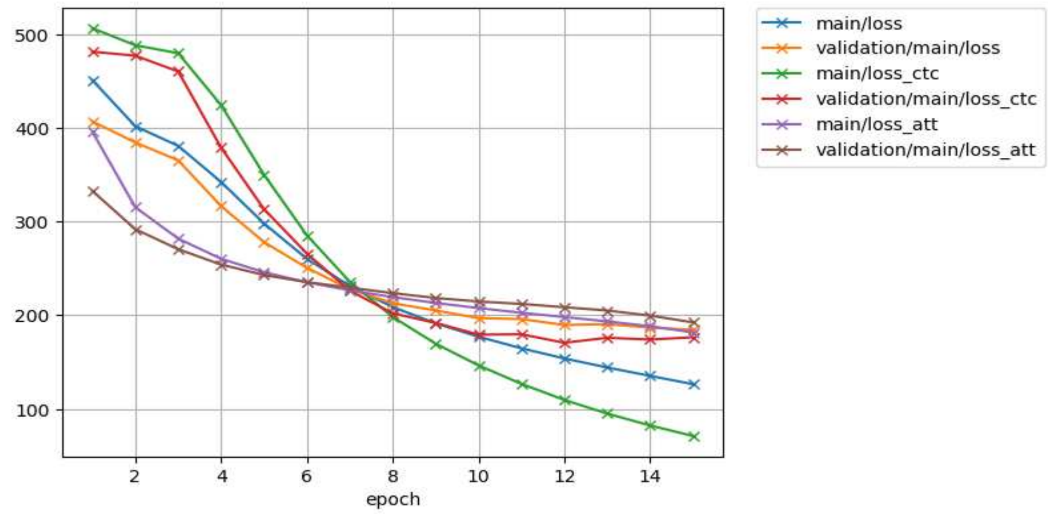

Figure 3.1: Loss vs Number of epochs

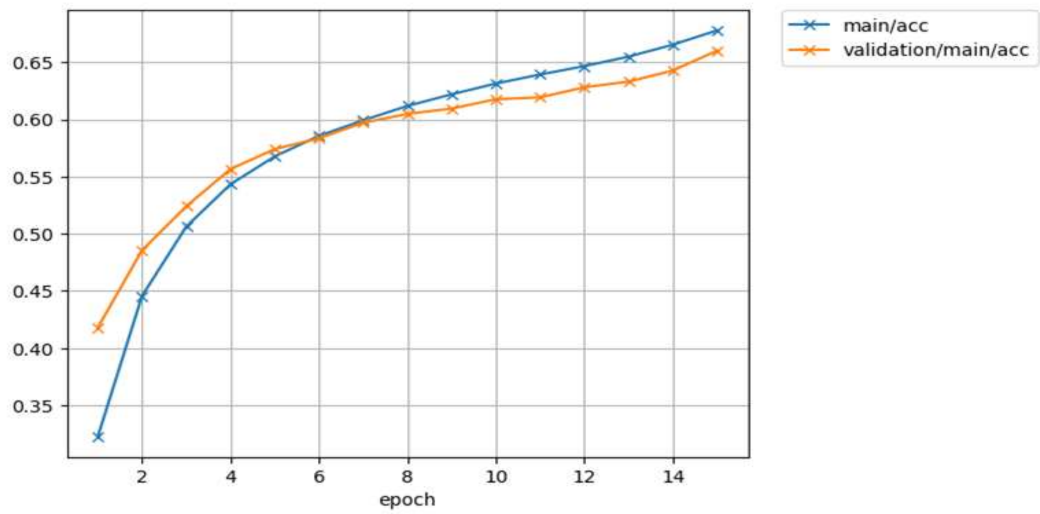

Figure 3.2: Accuracy vs Number of epochs

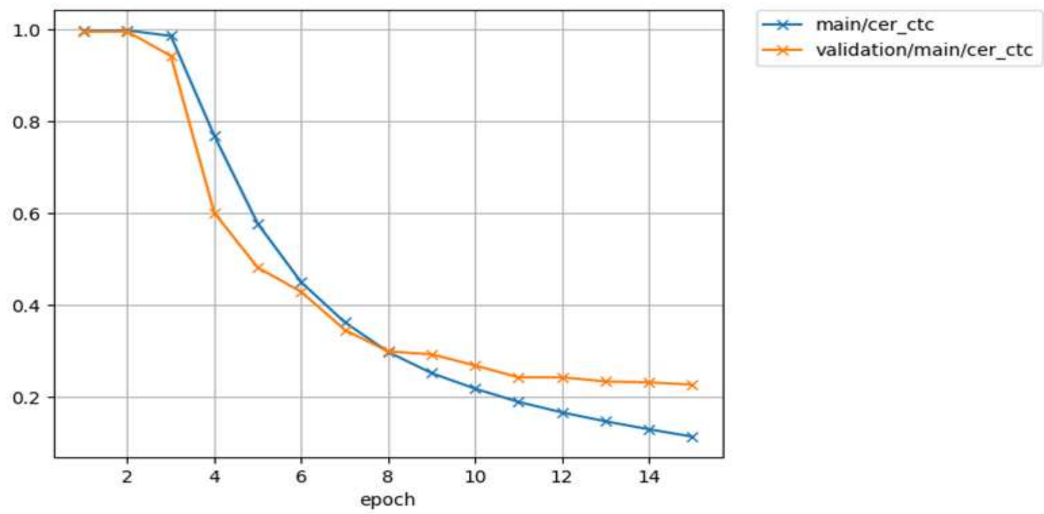

Figure 3.3: Character Error Rate vs Number of epochs

(1).eps

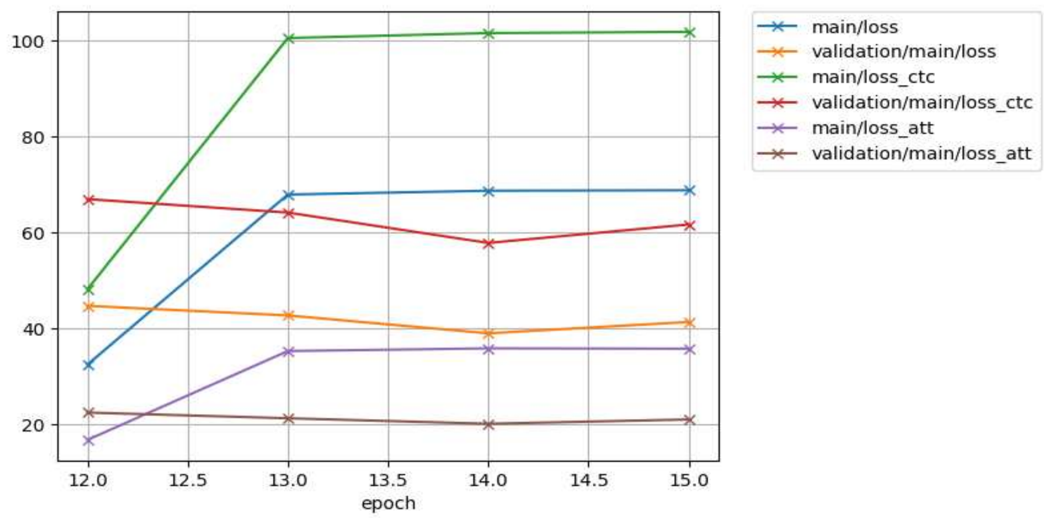

Figure 3.4: Loss vs Number of epochs

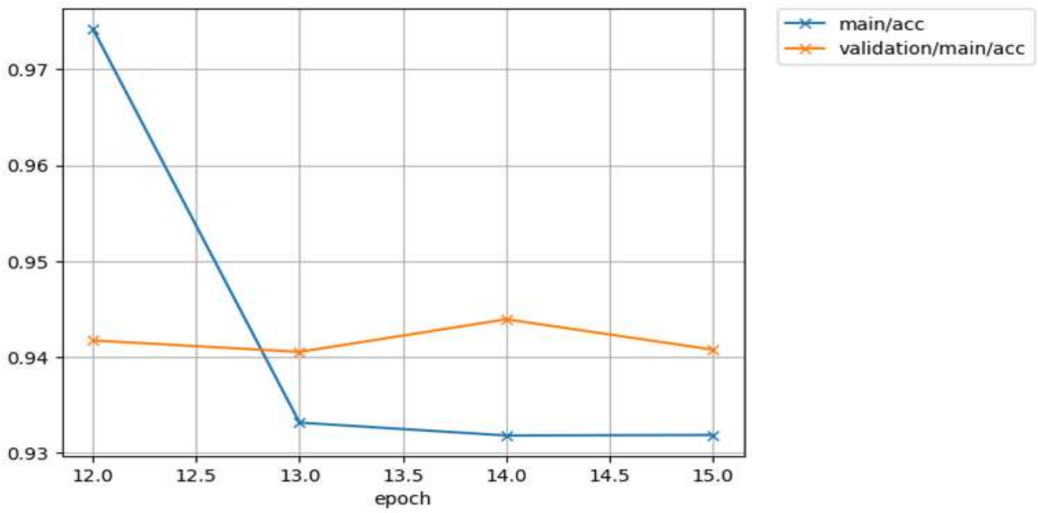

Figure 3.5: Accuracy vs Number of epochs

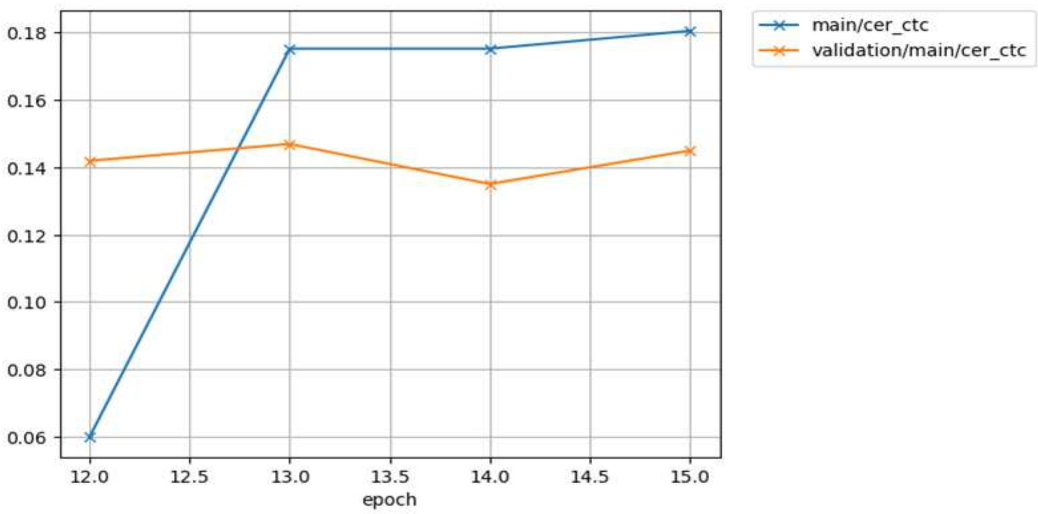

Figure 3.6: Character Error Rate vs Number of epochs

3.4 Observations

The following are the observations:

- As seen from the error plots above The training error continuously decreased with increase in the number of epochs for the case with 8 lectures because the data there is very less so it requires a large number of epochs to train that model.
- Contrary to which for the case with 400 lectures the validation loss is increasing after 11 epochs indicating over fitting which is expected.
- More the amount of data that we pour into the training of the rnn model in espnet. Better will be the word and character error rates as well that which is evident in a decrease of about 55.56% in the word error rates and a decrease of about 55.2% in the character error rate for the validation set than its predecessor model.

CHAPTER 4

Codes

The codes used in the project are written here.

4.1 run.sh

```
#!/bin/bash

# Copyright 2017 Johns Hopkins University (Shinji Watanabe)
#  Apache 2.0  (http://www.apache.org/licenses/LICENSE-2.0)

. ./path.sh
. ./cmd.sh


# general configuration
backend=pytorch
stage=4        # start from -1 if you need to start from data download
stop_stage=100
ngpu=0         # number of gpus ("0" uses cpu, otherwise use gpu)
debugmode=1
dumpdir=dump   # directory to dump full features
N=0            # number of minibatches to be used (mainly for
    debugging). "0" uses all minibatches.
verbose=0      # verbose option
resume=        # Resume the training from snapshot

# feature configuration
do_delta=false

# network architecture
# encoder related
etype=vggblstmp     # encoder architecture type
elayers=4
```

```
eunits=320
eprojs=320
subsample=1_2_2_1_1 # skip every n frame from input to nth layers
# decoder related
dlayers=1
dunits=300
# attention related
atype=location
aconv_chans=10
aconv_filts=100

# hybrid CTC/attention
mtlalpha=0.5

# minibatch related
batchsize=10 #30
maxlen_in=800  # if input length  > maxlen_in, batchsize is
    automatically reduced
maxlen_out=150 # if output length > maxlen_out, batchsize is
    automatically reduced

# optimization related
sortagrad=0 # Feed samples from shortest to longest ; -1: enabled for
     all epochs, 0: disabled, other: enabled for 'other' epochs
opt=adadelta
epochs=15
patience=3

# decoding parameter
beam_size=20
penalty=0
maxlenratio=0.0
minlenratio=0.0
ctc_weight=0.3
recog_model=model.acc.best # set a model to be used for decoding: '
    model.acc.best' or 'model.loss.best'

# scheduled sampling option
samp_prob=0.0

```

```
# data
voxforge=downloads # original data directory to be stored
lang=en # de, en, es, fr, it, nl, pt, ru

# exp tag
tag="" # tag for managing experiments.

. utils/parse_options.sh || exit 1;

. ./path.sh
. ./cmd.sh

# Set bash to 'debug' mode, it will exit on :
# -e 'error', -u 'undefined variable', -o ... 'error in pipeline', -x 'print commands',
set -e
set -u
set -o pipefail

train_set=tr_${lang}
train_dev=dt_${lang}
train_test=et_${lang}
recog_set="dt_${lang} et_${lang}"
#recog_set="dt_${lang}"
<<"over"
if [ ${stage} -le -1 ] && [ ${stop_stage} -ge -1 ]; then
    echo "stage -1: Data Download"
    local/getdata.sh ${lang} ${voxforge}
fi

if [ ${stage} -le 0 ] && [ ${stop_stage} -ge 0 ]; then
    ### Task dependent. You have to make data the following preparation part by yourself.
    ### But you can utilize Kaldi recipes in most cases
    echo "stage 0: Data Preparation"
    selected=${voxforge}/${lang}/extracted
    # Initial normalization of the data
    local/voxforge_data_prep.sh ${selected} ${lang}
    local/voxforge_format_data.sh ${lang}
fi
```

```
over

feat_tr_dir=${dumpdir}/${train_set}/delta${do_delta}; mkdir -p ${feat_tr_dir}
feat_dt_dir=${dumpdir}/${train_dev}/delta${do_delta}; mkdir -p ${feat_dt_dir}
if [ ${stage} -le 1 ] && [ ${stop_stage} -ge 1 ]; then
    ### Task dependent. You have to design training and dev sets by yourself.
    ### But you can utilize Kaldi recipes in most cases
    echo "stage 1: Feature Generation"
    fbankdir=fbank

    for x in ${train_dev} ${train_test} ${train_set}; do
  utils/fix_data_dir.sh data/${x}
        steps/make_fbank_pitch.sh --cmd "$train_cmd" --nj 10 --write_utt2num_frames true \
                data/${x} exp/make_fbank/${x} ${fbankdir}
    done
    # compute global CMVN
    compute-cmvn-stats scp:data/tr_${lang}/feats.scp data/tr_${lang}/cmvn.ark

    dump.sh --cmd "$train_cmd" --nj 10 --do_delta $do_delta \
        data/${train_set}/feats.scp data/${train_set}/cmvn.ark exp/dump_feats/train ${feat_tr_dir}
    dump.sh --cmd "$train_cmd" --nj 4 --do_delta $do_delta \
        data/${train_dev}/feats.scp data/${train_set}/cmvn.ark exp/dump_feats/dev ${feat_dt_dir}
    for rtask in ${recog_set}; do
        feat_recog_dir=${dumpdir}/${rtask}/delta${do_delta}; mkdir -p ${feat_recog_dir}
        dump.sh --cmd "$train_cmd" --nj 4 --do_delta $do_delta \
            data/${rtask}/feats.scp data/${train_set}/cmvn.ark exp/dump_feats/recog/${rtask} \
            ${feat_recog_dir}
    done
fi

dict=data/lang_1char/tr_${lang}_units.txt
```

```
echo "dictionary: ${dict}"
if [ ${stage} -le 2 ] && [ ${stop_stage} -ge 2 ]; then
    ### Task dependent. You have to check non-linguistic symbols used in the corpus.
    echo "stage 2: Dictionary and Json Data Preparation"
    mkdir -p data/lang_1char/
    echo "<unk> 1" > ${dict} # <unk> must be 1, 0 will be used for "blank" in CTC
    text2token.py -s 1 -n 1 data/tr_${lang}/text | cut -f 2- -d" " | tr " " "\n" \
    | sort | uniq | grep -v -e '^\s*$' | awk '{print $0 " " NR+1}' >> ${dict}
    wc -l ${dict}

    # make json labels
    data2json.sh --lang ${lang} --feat ${feat_tr_dir}/feats.scp \
         data/tr_${lang} ${dict} > ${feat_tr_dir}/data.json
    data2json.sh --lang ${lang} --feat ${feat_dt_dir}/feats.scp \
         data/dt_${lang} ${dict} > ${feat_dt_dir}/data.json
    for rtask in ${recog_set}; do
        feat_recog_dir=${dumpdir}/${rtask}/delta${do_delta}
        data2json.sh --feat ${feat_recog_dir}/feats.scp \
            data/${rtask} ${dict} > ${feat_recog_dir}/data.json
    done
fi



if [ -z ${tag} ]; then
    expname=${train_set}_${backend}_${etype}_e${elayers}_subsample${subsample}_unit${eunits}_proj${eprojs}_d${dlayers}_unit${dunits}_${atype}_aconvc${aconv_chans}_aconvf${aconv_filts}_mtlalpha${mtlalpha}_${opt}_sampprob${samp_prob}_bs${batchsize}_mli${maxlen_in}_mlo${maxlen_out}
    if ${do_delta}; then
        expname=${expname}_delta
    fi
else
    expname=${train_set}_${backend}_${tag}
fi
```

```
expdir=exp/${expname}
#mkdir -p ${expdir}

#exit 0
if [ ${stage} -le 3 ] && [ ${stop_stage} -ge 3 ]; then
    mkdir -p ${expdir}
    echo "stage 3: Network Training"
    ${cuda_cmd} --gpu ${ngpu} ${expdir}/train.log \
        asr_train.py \
        --ngpu ${ngpu} \
        --backend ${backend} \
        --outdir ${expdir}/results \
        --tensorboard-dir tensorboard/${expname} \
        --debugmode ${debugmode} \
        --dict ${dict} \
        --debugdir ${expdir} \
        --minibatches ${N} \
        --verbose ${verbose} \
        --resume ${resume} \
        --train-json ${feat_tr_dir}/data.json \
        --valid-json ${feat_dt_dir}/data.json \
        --etype ${etype} \
        --elayers ${elayers} \
        --eunits ${eunits} \
        --eprojs ${eprojs} \
        --subsample ${subsample} \
        --dlayers ${dlayers} \
        --dunits ${dunits} \
        --atype ${atype} \
        --aconv-chans ${aconv_chans} \
        --aconv-filts ${aconv_filts} \
        --mtlalpha ${mtlalpha} \
        --batch-size ${batchsize} \
        --maxlen-in ${maxlen_in} \
        --maxlen-out ${maxlen_out} \
        --opt ${opt} \
        --sortagrad ${sortagrad} \
        --sampling-probability ${samp_prob} \
        --epochs ${epochs} \
        --patience ${patience}
```

```
fi


#exit 0
recog_set="et_${lang}"
if [ ${stage} -le 4 ] && [ ${stop_stage} -ge 4 ]; then
    echo "stage 4: Decoding"
    nj=10

    pids=() # initialize pids
    for rtask in ${recog_set}; do
    (
        decode_dir=decode_${rtask}_beam${beam_size}_e${recog_model}_p${penalty}_len${minlenratio}-${maxlenratio}_ctcw${ctc_weight}
        feat_recog_dir=${dumpdir}/${rtask}/delta${do_delta}
        mkdir -p ${expdir}/${decode_dir}
        # split data
        splitjson.py --parts ${nj} ${feat_recog_dir}/data.json

        #### use CPU for decoding
        ngpu=0

        ${decode_cmd} JOB=1:${nj} ${expdir}/${decode_dir}/log/decode.JOB.log \
            asr_recog.py \
            --ngpu ${ngpu} \
            --backend ${backend} \
            --debugmode ${debugmode} \
            --verbose ${verbose} \
            --recog-json ${feat_recog_dir}/split${nj}utt/data.JOB.json \
            --result-label ${expdir}/${decode_dir}/data.JOB.json \
            --model ${expdir}/results/${recog_model}  \
            --beam-size ${beam_size} \
            --penalty ${penalty} \
            --maxlenratio ${maxlenratio} \
            --minlenratio ${minlenratio} \
            --ctc-weight ${ctc_weight}

        score_sclite.sh --wer true ${expdir}/${decode_dir} ${dict}
```

```
    ) &
    pids+=($!) # store background pids
    done
    i=0; for pid in "${pids[@]}"; do wait ${pid} || ((++i)); done
    [ ${i} -gt 0 ] && echo "$0: ${i} background jobs are failed." &&
   false
    echo "Finished"
fi
```

4.2 split_srt.py

```
import numpy as np
import os
import pydub
from pydub import AudioSegment
speaker="speakerid1"
uut_list=[]
current_directory = os.getcwd()
split_dir_name="splitwav2"     # temporary folder which contains .
   wavfiles downsampled to 16Khz and later saved to splitwav
final_directory = os.path.join(current_directory+"/"+split_dir_name)
#print(os.path.exists(final_directory))
if not os.path.exists(final_directory):
    os.makedirs(final_directory)
outFileName = os.getcwd()+"/" + "text"
outmp3FileName = os.getcwd() +"/"+ "wav.scp"
with open(outmp3FileName, "w") as f1:
    with open(outFileName, "w") as f:
        for mp3_files in os.listdir(os.getcwd()):
            if(".mp3" in mp3_files):
                sound = AudioSegment.from_mp3(mp3_files)

                file=mp3_files[:-4] + ".srt"
                timestamps=15.0 # duration of each audio file
                l=[]
                with open(file,"r", encoding="utf8") as fop:
                    l=fop.read().splitlines()
                    print(file)
                fop.close()
```

```
        dictionary={}# contains all info of start time, end time and text
        k=1
        file_length=len(l)
        for i in range(-1,file_length):
            temp=""
            if(l[i]=="" or i==-1):
                i+=2
                if(i<file_length):
                    temp_string=l[i]
                    start_time=int(temp_string[0:2])*60*60+int(temp_string[3:5])*60+int(temp_string[6:8])+int(temp_string[9:12])*0.001
                    loc=temp_string.find('>')
                    loc+=2
                    end_time=int(temp_string[loc+0:loc+2])*60*60+int(temp_string[loc+3:loc+5])*60+int(temp_string[loc+6:loc+8])+int(temp_string[loc+9:loc+12])*0.001
                    i+=1

                    while(l[i]!=""):
                        temp+=l[i] + " "
                        if(i+1 >= file_length):
                            break #for loop index exceeding
                        else:
                            i=i+1


                    temp_list_for_dict=[]
                    temp_list_for_dict.append(end_time)
                    temp_list_for_dict.append(temp)
                    dictionary[start_time]=temp_list_for_dict


        dict_keys=dictionary.keys()
        dict_len=len(dict_keys)
        temp_sum=0
        j=1
        print(len(sound))
        lenth=len(sound)
```

```
            temp_str=""
            start=None
            end=None
            time=None
            timeslots = []
            listofkeys = list(dictionary)
            for m in dict_keys:
                start=float(m)
                end=float(dictionary[m][0])
                time=end-start
                boolean = 0
                if(temp_sum+time>timestamps):
                    timeslots.append(float(m))
                    l=len(timeslots)

                    # if(l==1):
                    #     outFileName = os.getcwd() + "\Western_philosophy" + "_" + str(0) + "_" + str(timeslots[l-1]) + "_.txt"
                    # else:
                    #     outFileName = os.getcwd() + "\Western_philosophy" + "_"+ str(timeslots[l-2]) +"_"+ str(timeslots[l-1]) + "_.txt"
                    #outFile=open(outFileName, "w")
                    f.write(mp3_files[:-4] + "_")
                    if(l==1):
                        f.write(str(listofkeys[0]) + "_" + str(timeslots[l-1]) + "        ") #Utterance ID
                        uut_list.append(mp3_files[:-4] + "_"+str(listofkeys[0]) + "_" + str(timeslots[l-1]))
                    else:
                        f.write(str(timeslots[l-2]) + "_" + str(timeslots[l-1]) + "        ")#Utterance ID
                        uut_list.append(mp3_files[:-4] + "_"+str(timeslots[l-2]) + "_" + str(timeslots[l-1]))
                    f.write(temp_str.lower())# text
                    f.write("\n")
                    temp_str=dictionary[m][1]
                    temp_sum=time
                    j+=1
```

```
                f1.write(mp3_files[:-4] + "_")
                if(l==1): # because we know only end time. we didnt know start time. It can be pulled from listofkeys of dictionary
                    temp_audio_file=sound[float(listofkeys[0])*1000:float(timeslots[l-1])*1000]# spliting audio according to start and end time
                    temp_audio_file.export(final_directory+"/"+"Western_philosophy"+ "_" + mp3_files[:-4] + "_" + str(listofkeys[0]) + "_" + str(timeslots[l-1]) + "_.wav",format="wav")
                    f1.write(str(listofkeys[0]) + "_" + str(timeslots[l-1]) + "        ")#for wav.scp
                    f1.write(os.getcwd()+"/"+split_dir_name+"/"+ "Western_philosophy"+ "_" + mp3_files[:-4] + "_" + str(listofkeys[0]) + "_" + str(timeslots[l-1]) + "_.wav")#audio paths for wav.scp
                    f1.write("\n")
                else:
                    temp_audio_file=sound[float(timeslots[l-2])*1000:float(timeslots[l-1])*1000]
                    temp_audio_file.export(final_directory+"/"+"Western_philosophy" + "_"+ mp3_files[:-4] + "_" +str(timeslots[l-2]) + "_" + str(timeslots[l-1]) + "_.wav",format="wav")
                    f1.write(str(timeslots[l-2]) + "_" + str(timeslots[l-1]) + "        ")
                    f1.write(os.getcwd() +"/"+split_dir_name+"/"+ "Western_philosophy" + "_"+ mp3_files[:-4] + "_" +str(timeslots[l-2]) + "_" + str(timeslots[l-1]) + "_.wav")
                    f1.write("\n")
            else:
                if(boolean == 0):
                    temp_str+=dictionary[m][1]
                    temp_sum+=time
                    boolean = 1
                else:
                    temp_str+=" " + dictionary[m][1]
                    temp_sum+=time
        f.write(mp3_files[:-4] + "_")#for last segment
        f1.write(mp3_files[:-4] + "_")# for last split
        if(l==1):
```

```
                    f.write(str(listofkeys[0]) + "_" + str(listofkeys[len(listofkeys)-1]) + "     ")
                    temp_audio_file=sound[float(listofkeys[0])*1000:float(listofkeys[len(listofkeys)-1])*1000]
                    temp_audio_file.export(final_directory+"/"+"Western_philosophy"+ "_" + mp3_files[:-4] + "_" + str(listofkeys[0]) + "_" + str(listofkeys[len(listofkeys)-1]) + "_.wav",format="wav")
                    f1.write(str(listofkeys[0]) + "_" + str(listofkeys[len(listofkeys)-1]) + "     ")
                    f1.write(os.getcwd()+"/"+split_dir_name+"/" + "Western_philosophy"+ "_" + mp3_files[:-4] + "_" + str(listofkeys[0]) + "_" + str(listofkeys[len(listofkeys)-1]) + "_.wav")
                    f1.write("\n")
                    uut_list.append(mp3_files[:-4] + "_"+str(listofkeys[0]) + "_" + str(listofkeys[len(listofkeys)-1]))
                else:
                    f.write(str(timeslots[l-1]) + "_" + str(listofkeys[len(listofkeys)-1]) + "     ")
                    temp_audio_file=sound[float(timeslots[l-1])*1000:float(listofkeys[len(listofkeys)-1])*1000]
                    temp_audio_file.export(final_directory+"/"+"Western_philosophy" + "_"+ mp3_files[:-4] + "_" +str(timeslots[l-1]) + "_" + str(listofkeys[len(listofkeys)-1]) + "_.wav",format="wav")
                    f1.write(str(timeslots[l-1]) + "_" + str(listofkeys[len(listofkeys)-1]) + "     ")
                    f1.write(os.getcwd() +"/"+split_dir_name+"/"+ "Western_philosophy" + "_"+ mp3_files[:-4] + "_" +str(timeslots[l-1]) + "_" + str(listofkeys[len(listofkeys)-1]) + "_.wav")
                    f1.write("\n")
                    uut_list.append(mp3_files[:-4] + "_"+str(timeslots[l-1]) + "_" + str(listofkeys[len(listofkeys)-1]))
                f.write(temp_str.lower())
                f.write("\n")
    f.close()
f1.close
with open(os.getcwd()+"/"+"utt2spk","w") as uf:
    for i in uut_list:
        uf.write(i+" "+speaker+"\n")
```

```
    uf.close()
with open(os.getcwd()+"/"+"spk2utt","w") as spf:
    spf.write(speaker+" ")
    for i in uut_list:
        spf.write(i+" ")
    spf.close()
```

4.3 resample.sh

This script calls the above split_srt.py and downsamples all audio files to $16Khz$.

```
#!/bin/bash
mkdir -p splitwav
python split_srt.py
for entry in `ls splitwav2/`; do
    echo ${entry}
  sox splitwav2/${entry} -r 16000 splitwav/${entry} #downsample the .wav files in splitwav2 created by split_srt.py and put them in splitwav
done
rm -r splitwav2
```

LIST OF PAPERS BASED ON THESIS

1. Takaaki Hori, Jaejin Cho, Shinji Watanabe END-TO-END SPEECH RECOGNITION WITH WORD-BASED RNN LANGUAGE MODELS *Journal*, 1, (Aug, 2018).

2. Thomas Zenkel, Matthias Sperber, Jan Niehues, Markus Müller, Ngoc-Quan Pham, Sebastian Stüker, Alex Waibel Open Source Toolkit for Speech to Text Translation (Oct 2018)