AUTOMATIC SPEECH RECOGNITION FOR INDIC LANGUAGES

Raghav Karan
MENTOR: DR. Vipul Arora

INTRODUCTION

Automatic Speech Recognition (ASR) technology has witnessed significant advancements in recent years, revolutionizing the way we interact with machines and opening up new avenues for communication and accessibility. ASR systems play a pivotal role in converting spoken language into written text, enabling seamless transcription, documentation, and analysis of verbal information across various languages and domains. The increasing demand for accurate and efficient ASR solutions has driven extensive research and development efforts, resulting in notable progress in areas such as voice assistants, virtual agents, call centers, language translation, and data analysis. ASR has become an indispensable tool in bridging communication gaps, improving accessibility for individuals with hearing impairments, and enhancing human-machine interactions.
This research project delved into the domain of ASR for Indic languages, namely Hindi and two low-resource languages-Bhojpuri and Bengali. This discussion aims to shed light upon the work undertaken in Madhav Lab on developing ASR models and tackling challenges like Long-form Audio, test-time adaptation and active learning.

OBJECTIVE

The title of the project was initially decided upon as Automatic speech recognition (ASR) for Indian English, but early in the duration of the project, the focus shifted to ASR for Hindi. The project was about developing an ASR model for Hindi language for an external organization. The ultimate purpose of the model was to provide transcription for audios of extremely long durations. The project has been going on for some time with Professor Vipul Arora and Prasar Bharati, India's state-owned public broadcasting. Prasar Bharati has a lot of spoken data as they have been offering their services for a long time now. Their extensive array of network includes All India Radio-Akashvani and Doordarshan among others. They require an ASR model for the extensive transcription of data they have, which is mostly long form audios, which posed another challenge. They also require audio subtitling, and hence generating audio transcriptions along with time stamps was required. More details of the project will follow later. After working and completing tasks at hand with the mentor's team on Prasar Bharati project, the focus shifted to a global ASR challenge, MADASR'23, organized by IEEE-ASRU (Workshop on Automatic speech recognition and Understanding) and IISC Bangaluru. The challenge involved adapting ASR models for low resource Indic Languages-Bhojpuri and Bengali. The challenge involved a lot of different aspects that were worked on.

ASR MODELS

A **conformer** model is a modern architecture used in Automatic Speech Recognition (ASR) systems that combines convolutional neural networks (CNNs) and self-attention mechanisms. It excels in capturing both local and global dependencies in speech data. The model leverages convolutional layers to extract local acoustic features and employs self-attention to capture long-range dependencies, allowing it to consider context from past and future speech frames. With the help of a feed-forward module for refining representations, the conformer model achieves state-of-the-art performance by effectively mapping input speech features to corresponding textual representations. Its ability to handle long sequences and capture intricate relationships has made it a highly effective and sought-after approach in ASR research.

The model utilizes **Connectionist-Temporal-Classification**. Connectionist Temporal Classification (CTC) is a popular framework used in Automatic Speech Recognition (ASR) for training and decoding sequence-to-sequence models. It addresses the challenge of aligning variable-length input speech features with output transcriptions. CTC introduces a special "blank" label and allows multiple labels to collapse into a single output label, enabling the model to learn alignments without explicit alignment information. During training, the CTC loss function measures the difference between predicted and target sequences, training the model to optimize this loss and indirectly learn the alignment. During decoding, the CTC algorithm converts output probabilities into the most likely transcription. CTC offers flexibility, handling variable-length sequences, and accommodating various label sets. However, it may produce repeated or inaccurate predictions, which can be mitigated using language models or post-processing techniques.

Hindi

Trained with the extensive data provided by Prasar Bharati, which included audios ranging from news segments from Doordarshan to Prime Minister Shri Narendra Modi's Mann Ki Baat.

Bhojpuri

Bhojpuri language has the same script as Hindi, that is, Devnagari. Therefore, pre-existing Hindi models that could be fine-tuned with Bhojpuri data was the first approach the team thought of.

Fine-tuning a pre-trained model involves adapting an already trained model like wav2vec, conformer, etc., to a specific task or domain by further training it on a smaller, task-specific dataset. The process begins with pre-training the model on a large dataset to learn general patterns and representations. In our case, this was Hindi. Pre-trained Hindi models are available readily. Then, a task-specific dataset is collected or created, and the pre-trained model's weights serve as a starting point. The model is further trained on the novel dataset, allowing it to adapt to the target task by learning new data centric features. Fine-tuning reduces the need for extensive labeled data and can improve performance on the target task by leveraging the knowledge gained during pre-training.

Fine-tuning Hindi pre-trained models were experimented with, but the best results were obtained from training a conformed model from scratch.

Bengali

Bengali posed a challenge due to its script, which is very difficult to be adapted by a Hindi pre-trained model. Hence, transliteration to English was explored. English pre-trained model are available abundance on the internet and thus can be used for applications readily. But the transliteration results were unsatisfactory.

Conformer medium model used, and the training was done from scratch.

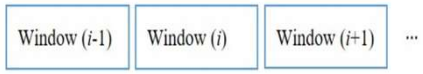
LONG-FORM AUDIO

Models are generally trained on audios of small duration, due to limitations of memory and computations. In our case, the model was trained on audios that were 16 seconds of duration each. Also, the size of audios the models can process in a single go is limited by the system configuration, and thus long audios require some extra efforts in this regard.

Some project solutions are listed below. The results were based on a few 30-minute Mann ki Baat audios of our respected PM and 15-minute news audios. News audios had lots of BGM, but Mann ki Baat did not. Prasar Bharati supplied all audio.

Method 1: Hard Chunking

In this method, the long audio is broken into smaller segments or windows of fixed length. This length is kept at a suitable value less than the maximum length the model is capable of handling without the GPU going out of memory. Individual transcriptions are simply concatenated together with space in between.



Method 2: Chunking with strides

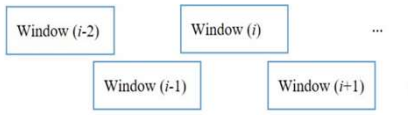
The input audio waveform is broken into smaller chunks with some overlap. Individual audios are fed into the model and frame-by-frame transcriptions are obtained. The transcriptions are sliced from the starting and the ending to delete the overlapping portions. They are then collapsed and stitched into a single transcription.

Method 3: Partially overlapped inference and stitching using edit distance with modified operation costs over individual predictions.

Input audio is chunked into small waveforms with some overlap. The uncollapsed predictions for each are obtained from the model. The predictions are then stitched together via edit-distance based score matrix. For stitching them together, a list of individual transcriptions is iterated repeatedly. In each iteration, pairwise alignments of transcriptions is done. For e.g., if the list contains 10 individual chunk transcriptions, in the first iteration, we merge the pairs (1,2),(3,4,...(9,10) with in place replacement. In the next iteration we repeat the same steps for the novel merged transcripts.

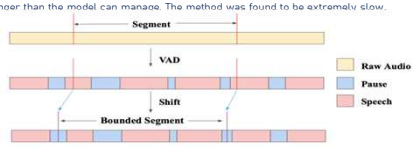
Method 4: Partially overlapped inference and stitching using edit distance with modified operation costs only over the overlapped window.

This method is like above, the only difference is that the score matrix is generated for the overlapped portions only rather than the entire chunk transcription. The non overlapped portions and overlapped portions are concatenated together. The previous method, in principle, is superior to this as it has more context, but this method reduces the time complexity over the previous, though still higher than some, enabling us to obtain faster inferences.



Method 5: Voice Activity Detection (VAD) overlapping inference

The input audio is passed into a VAD model which outputs timestamps of non-silent regions. These timestamps are used to generate chunks which begin and end at the middle of silent regions. The individual transcriptions are obtained and concatenated together with spaces, to represent silence, in between. This method does not help with audios that contain background music, as the silences are occasional, and the individual chunks even with VAD are mostly longer than the model can manage. The method was found to be extremely slow.



Method 6: Chunking using CTC decoder outputs.

This method discovers silences with the help of CTC decoder outputs. We begin at the start of the audio, and hard chunk it at a particular duration. This hard chunk is fed into the model and the uncollapsed frame-by-frame predictions are obtained from the CTC decoder. The chunk prediction is then iterated from the end to search for regions of long silence represented by continuous blank tokens. We stop at the first long silence we find. This silence region is then used to modify the current hard chunk by terminating it at the center of found region of silence. The new end point is noted which would serve as the starting point of the next hard chunk in the next iteration. The inferences are obtained for the modified chunk again.

The above snap is a CTC decoder's frame by frame prediction. The asterixis represent blank tokens treated as pauses.

Method	Word Error Rate	
	Audio without BGM (%)	Audio with BGM (%)
Hard Chunking	22.16	41.53
Chunking with strides	21.35	42.23
Partial Overlapping		
(10% overlap)	22.4	46.67
(50% overlap)	20.76	39.81
VAD based silence	25.01	-
CTC based silence	17.68	33.67

INTEGRATION WITH BEAM SEARCH

The method was integrated with beam search and language model which reduced the errors remarkably. Beam search is a decoding algorithm commonly used in sequence generation tasks, including Automatic Speech Recognition (ASR). It is used to find the most probable sequence of output tokens given a sequence of input tokens, such as converting an audio waveform into its corresponding text transcription.

The beam search algorithm explores multiple possible paths through the output space in a systematic manner. Instead of considering only the top-ranked hypothesis at each decoding step, beam search maintains a set of top-K hypotheses, known as the beam width. The beam width determines the number of paths or hypotheses that are considered simultaneously during decoding.

Without BGM
WER with Beam search: 11%
WER with Beam search and language model: 7%

Challenge with beam search: It required SoftMax probability matrix as input, as it must go through various probable sequences . So, first the silence was found on uncollapsed indices, then the particular dimension of the probability matrix was sliced accordingly, which was subsequently provided to beam search.

Generating timestamps

The next task was to generate chunk transcriptions with time stamps. The intricate dependence of frame numbers and the time duration were explored and utilized to generate the time stamps. The raw audio is used by the model as a mel-spectrogram which has generally 16000 (called the sample rate) samples generated per second, which is subsequently resampled by convolution layers by a factor of 4. The samples are ultimately condensed into frames. The time stamps were obtained with the help of chunk starting and ending points obtained from CTC decoder's outputs. We find the sample number the chunk begins and ends at, which is divided by the sample rate to obtain the time stamp in seconds. Time stamps have many applications, the most prominent being subtitles srt files

ASRU MADASR CHALLENGE

MADASR'23 (**M**odel **AD**aptation for **ASR** in low-resource Indian languages): The challenge was to adapt existing ASR models for low resource Indic languages: Bhojpuri and Bengali, which have much less data available to train.

Transliteration experiments

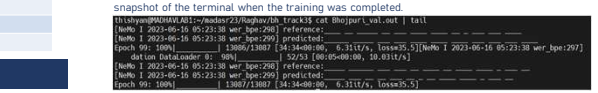
Initial approach: the team decided to utilize a pre-trained model and fine-tune it for the new language. Bhojpuri and Hindi both use Devnagari script and thus we can directly use a Hindi pre-trained model and fine- tune it with limited Bhojpuri data. However, Bengali has an altogether different script. We explored transliteration to English so as to use an English pre-trained model to fine tune with transliterated Bengali data. Different libraries were experimented with, and results are briefly specified below.

1.	**AI4bharat transliteration tool:**	WER: 33.4 ; CER=11.5 ; Time=1.46s/it
2.	**Polyglot:**	Could not work as it required many dependencies in the system and issues with that could not be resolved.
3.	**Google transliterate:**	WER: 30.7 ; CER=11.3 ; Time=4s/it
4.	**Indicnlp:**	Works only for Indic to Indic, does not support Latin transliteration.

ASR for Bhojpuri: Fine-tuning pre-trained model

Nemo Pre-trained model

Fine-tuning Hindi pre-trained model for Bhojpuri: Exploration about pre-trained models led us to models provided by Nvidia's Nemo toolkit . A Hindi conformer model trained on 128 sub-word piece tokens was discovered. The Bhojpuri dataset was prepared as per the requirements of the model. The entire data was made into manifest files (Json), that contained the path to audio, duration, and the reference transcript. The vocabulary is supposed to be updated as per the new dataset, and thus 128 token vocabulary was decided upon as that could use the weights of the model directly. The convergence took 3 days for 100 epochs of training on the data. Below is a snapshot of the terminal when the training was completed.



The team later decided to train a conformer model previously used for Hindi language from scratch and tally the results. The training took a few days, but the results were better than nemo pre-trained model and WER was 25% on validation dataset

Track	Bhojpuri		Bengali	
	WER	CER	WER	CER
Greedy decoder	27.04	8.98	26.96	7.89
Greedy decoder +Beam Search	26.89	9.72	26.40	9.33
Greedy decoder +Beam Search + LM	20.91	7.78	12.29	5.27

Test-time-adaptation

After the model was trained, the team decided to experiment with test-time adaptation (TTA) to obtain reliable results with the test data.

Source free single utterance TTA (SUTA) : This method aims to adapt a trained model to each new audio in the test set. It generate the labels for the test audio and calculate entropy loss, class confusion, and try to minimize these losses. Entropy minimization sharpens the class distribution and minimum class confusion reduces correlations between classes. Only the layer normalization and encoder weights are updated for better results

Active learning

The next task involved generating baseline results for a new method being developed in Lab for active learning in automatic speech recognition. An existing active learning strategy was implemented to obtain results that will be used to compare with the novel method developed in lab. The method uses path probabilities of the generated inference for each audio in the dataset. The confidence scores or path probabilities of the predictions are calculated by multiplying each individual probabilities of frame predictions.

$$ppath = \frac{\log(P(y|x))}{len(y)}$$

$$P(y|x) = \prod_{y_i \in Y} P(y_i|x)$$

y_i's are the SoftMax probabilities of the char chosen for a particular frame. We multiply them all together to get the probability of the sequence that has been selected for the current audio. The inferences model is less confident about would have lesser total probabilities as the individual probabilities will be distributed among the classes. We then selected a specific number of audios the model is the least confident about. We get this data annotated and retrain the model. This is continued until satisfactory results are achieved

ACKNOWLEDGEMENT

I would like to express my gratitude to Surge for including me in their programme in the first place. I would like to take this opportunity to express my thanks and respect to my mentor, Professor Vipul Arora, for making this opportunity possible for me. My time spent in the laboratory has provided me with an extremely enlightening experience overall. The trip was chock full of educational opportunities. I would also like to express my gratitude towards Rathna Ma'am, Post-Doc Research fellow and Thishyan Raj, MSR student at the Madhav Lab. They have provided me with guidance and mentoring throughout the entirety of the project.

REFRENCES

Tae Gyoon Kang et al., "PARTIALLY OVERLAPPED INFERENCE FOR LONG-FORM SPEECH RECOGNITION," ICASSP 2021.

Jinhan Wang et al., "VADOI: VOICE-ACTIVITY-DETECTION OVERLAPPING INFERENCE FOR END-TO-END," ICASSP 2022 .

G. L. S. &. L. H. Lin, "Listen, Adapt, Better WER: Source-free Single-utterance Test-time Adaptation for Automatic Speech Recognition," ArXiv. /abs/2203.14222.

Jihwan Bang et al., "BOOSTING ACTIVE LEARNING FOR SPEECH RECOGNITION WITH NOISY PSEUDO-LABELED SAMPLES," arXiv:2006.11021v2.

MADHAV lab
Machine Analysis of Data for Human Audition and Vision

Name: Raghav Karan
Application number: 2330001
Mentor: Dr. Vipul Arora
Mentor Department: Electrical Engineering

www.PosterPresentations.com