# Abstract

This project presents the usage of Automatic Speech Recognition System (ASR) for Nepali language as a low-resource language. Traditional ASR algorithms like DeepSpeech required huge amount of data for training the ASR engine to a reasonable accuracy resulting in training low-resource language like Nepali to be difficult. This paper uses a new algorithm Unsupervised Cross-lingual Representation Learning for Speech Recognition, named Wav2Vec2.0. The algorithm allows for pretraining on unlabeled data. This allows the usage of unlabeled data from any language to improve the accuracy of the ASR engine which are present in abundance. This paper describes the usage of Wav2Vec2 in Nepali language for creating a base model using freely available data on the Internet.

# Introduction

Automatic Speech Recognition is very important for the human-computer interaction. This will allow humans to communicate with computers using the means of communication that is most common and easy to use, voice. But with Deep Learning models, the need for training data i.e., labeled audio data is very high with the amount of data required being in hundreds and even thousands for different accents, linguistical approach and difficult languages. This need harms the diversity in the availability of ASR engines for low-resource language.

There are more than 7000 languages spoken around the world (Eberhard, Simons and Fennig, 2021). While some of them are widely spoken and many people know about them, many are less spoken. Among these, only about 500 languages would have an ASR engine developed which has a decent enough accuracy. This is a result of the low availability of the labeled audio data. The remaining 6000+ languages are affected due to the requirements of the current widely used automatic speech recognition models.

The main goal of the project is to use and validate the Wav2Vec2 model released by the Facebook AI team. For the implementation and evaluation of the model, Nepali language, a low-resource language, is used. With proper validation and implementation of the project, the project can be further extended into other languages with very less customization required. The Wav2Vec2 model, if works as reported by the Facebook AI team, would greatly revolutionize the field of ASR tech, and would help in the generation and creation of ASR models for many low-resources languages.

The primary aim of the project is the development of an Automatic Speech Recognition technology (Speech-To-Text) technology for low-resource languages, Nepali languages for the implementation and evaluation. The project would allow for the usage of speech technologies for low resource languages and would allow for NLP (Natural Language Processing) development along with Voice Activated AIs using the Nepali language/ Devanagari Script.

The primary objectives of the project are listed below:

* Research on the literatures and technologies used currently for the Automatic Speech Recognition and their implementation
* Creation of an ASR engine for Nepali languages and Speech-to-Text technologies in Devanagari Script
* Exploration of the technology for future studies and progress like Natural Language Processing in Nepali Speech

First and foremost, the research question of the lack of ASR product was seen through research of ASR product and their viability in different languages. When going through even the state-of-the-art ASR engines like DeepSpeech, it was noted that there were not a lot of languages, especially low-resource languages like Nepali supported in these frameworks or even if there was a viable product, the accuracy was very bad. This led to the research for the techniques for the development of an accurate enough ASR engine for these low-resource languages. The data for the development of the product i.e., the ASR engine was collected from online sources, namely OpenSLR. While other means of data collection like recording from the people, this idea was scraped due to the time and budget constraints since collecting the audio recording, segmenting the data, and finally transcribing the data would be a very long process and would cost expensive money.

The product from the research would be a template, a type of framework that can be used in different languages with just the provided data being different and some optimizations if required. Through this, even low-resource languages would be able to develop an ASR engine which would allow for further analysis and processing of the language (for example: Natural Language Processing).

# Theoretical Framework

## Background

Over the last few decades, automatic speech recognition, also known as speech-to-text (STT) has been a widely researched topic as it is one of the important communications means for human-computer interaction. Development of the automatic speech recognition field has been rapid over the last 5 – 10 years due to the development of newer computational hardware and architecture, newer and efficient algorithms. Since speech is the primary communication means for human interaction, interaction between human and computer would be much easier if this mode of interaction were possible where commands and actions can be carried out by computers using human speech as the primary mode of input. Automatic Speech Recognition, ASR in short, is the field in computing that aims for the transcription of utterance in an audio using the speech waveforms. While speech understanding goes a little further by using the transcribed text as the base for Natural Language Processing to understand the intent and meaning of the spoken utterances, ASR is an important part of understanding and parsing the speaker’s command.

The working principle of ASR depends on the parsing of the speech signal from the audio, whether it be a recorded audio or through the microphone and converting these waveforms into text sequence according to the trained data. The whole process consists of parsing the audio to acoustic waveforms and using linguistic parsing and processing, converting them to set of words. Since the first attempt of converting speech signal based on conversation into sequence of phoneme-like units in 1950, automatic speech recognition has come along a long way. While the automatic speech recognition showed positive results for the first time in 1970 with the usage of general pattern matching techniques, the algorithms and computing mechanism for automatic speech recognition has seen rapid advancement over the last decade (Singh, 2016). Along with the development in the field of automatic speech recognition, this field has garnered heavy attention with possible usage in various other fields of education, health, media and so on.

The accuracy and quality of the ASR engine is heavily reliant on the usage of quality audio data used for the training of the engine. With the multilingual and multicultural background of the people around the globe, training the “perfect” ASR engine is almost impossible from the current technology and algorithms. Along with this, even in the same language, there are multiple accents spoken by people around the world which heavily affects the quality and accuracy of the transcribed text using the ASR engines. For example, English is the language with the most speakers around the globe. We can see the differences in the different accents spoken by people of different countries and locales like the U.S., the U.K., Asia, Africa and so on. This alone presents a huge challenge for the training of a proper ASR engine. Since training data plays a huge role for the better training of the ASR engine, the amount of training data present for the tuning of the engine also plays a huge role. This results in low-resource languages like Nepali, where the amount of quality labelled data for the purpose of training ASR engine is very difficult and costly for collection.

This research presents a new approach for the training of ASR engine for low resource language, in this case Nepali language. The ASR engine used in this case is based on the algorithm presented by Baevski et al. on 22 Oct 2020 namely wav2vec2, as a successor to wav2vec. The algorithm uses self-supervised learning from the raw audio representation and fine-tuning the model using labeled data to train the model to predict the correct speech unit as required. This allows the usage of raw audio data from other languages and fine tuning the model for required specific model. Using the wav2vec2 algorithm, with just 10 minutes of transcribed audio data with model trained on 53K hours of unlabeled speech, the speech recognition model had an WER (Word Error Rate) of 8.6% on speech with noises and 5.2% on clean speech using the standard LibriSpeech English Benchmark. (Baevski, Conneau and Auli, 2020). This is a revolution in the field of ASR and would allow for the usage of speech recognition models in more variety of languages including low-resource languages, languages with multiple dialects and multiple domain usage of the languages which would have required huge amounts of quality transcribed data if traditional algorithms were used to provide an acceptable accuracy.

Wav2Vec 2.0 is wholly based on the usage of self-supervised learning on the raw representations of the unlabeled speech and the finetuning through the usage of labeled data leading to an overall improvement in the usage of ASR, widening its usage domains and even helping to improve the current existing systems.

### History of Automatic Speech Recognition

The history of Automatic Speech Recognition begins with the development of telephone by Alexander Graham Bell which processed the audio signals/ waves into electrical impulses (The First Telephone Call, 2021) . Without the invention of the telephone, processing audio signals would have been neigh impossible until another invention for that purpose was made.

The first actual Automatic Speech Recognition engine was developed in 1952 by Bell Labs. The engine could transcribe simple numbers and was named Audrey (Manjutha, Gracy, Subashini and Krishnaveni, 2021). Any actual breakthrough was not made until 1962, a decade later, when IBM developed an ASR engine which could recognize and transcribe 16 English words. Later, a collaboration of the major powers; Soviet Union, United States, England, and Japan, developed an ASR engine which could recognize 4 vowels and 9 consonants from the English alphabets. Between 1971 and 1976, Carnegie Mellon developed an automatic speech recognition engine named “Harpy” which could understand and recognize 1011 words. All the technologies and hardware built to that time was non-commercial with the first commercial speech recognition technology being developed by Threshold Technology and Bell Laboratories.

The next breakthrough in the field of ASR was the development of a new statistical method, Hidden Markov Models (HMM) developed in 1970 (O’Shaughnessy, 2008). The use of HMM was based on probability functions to determine the words for transcription through feeding phonemes, the smallest unit of sound for a word, into a program using HMM and transcribing the audio with the highest probable word. Later, the model was optimized through noise reduction techniques during data processing and the usage of beam search language models, model which finds the best fit for the target word using the before and after words of the required word. The full process was called “tri-gram” model and more than 80% of the current ASR engine and technologies are based on the refined version of these “tri-gram” models (Lam, 2021).

The next generation of the Automatic Speech Recognition was the introduction of neural network engines for model training . This was first started during the late 1980s. The usage of neural network greatly improved the ASR models accuracy with better phoneme differentiation and better word prediction. However, this came with the need of higher processing power requirements for audio parsing and transcribing. This caused businesses and enterprises with need for huge amount of audio data processing to have higher cost. So, businesses were needed to get the balance between accuracy and budget for proper and stable ASR engine usage.

While many believed the neural network to be a key for the new ASR, with the advent of big data, higher computing performances, GPU (Graphical Processing Unit) processing power, a new ASR was introduced, End-to-End Deep Learning ASR (Wang, Wang and Lv, 2019). This new ASR engine used the advantages of deep learning to “train” the engine to understand various transcribed/ labeled data and use this model for transcribing audio data fed into it. With higher amount of labeled data, the accuracy of the ASR engine increased. Most of the modern ASR models use this method like DeepSpeech.

But this, in turn, increased the need for the amount of labeled audio data for having a decent enough accuracy for the ASR engines. While this was not a problem for common spoken languages like English, Spanish, Chinese and so on, languages with very low amount of labeled data suffered an accuracy hit when training, like Nepali language. On September 2019, a group of Facebook AI researchers released a new training method for ASR models, Wav2Vec (Schneider, Baevski, Collobert and Auli, 2021). They used unsupervised pre-training for the speech recognition through the training on the raw representations of raw audio data. Wav2Vec is pre-trained on the large amounts of unlabeled audio data and then the resulting audio representations are used for improving the acoustic model training. The pre-training is applied to a multi-layer CNN (Concolutional Neural Network) and optimized through the use of noise contrastive binary classification task. Then, the model is fine-tuned using accurate labeled data. This greatly decreased the amount of data required for the model training while providing significantly greater accuracy as compared to the best in class of current ASR models, DeepSpeech 2 (Schneider, Baevski, Collobert and Auli, 2020). The next evolution of Wav2Vec was the introduction of Wav2Vec 2.0 (Baevski, Conneau and Auli, 2020). This allowed for the usage of self-supervised learning during the pre-training of the models of unlabeled data through the masking of speech input in the latent speech and solving the task defined over quantization of the latent representations learned together during pre-training.

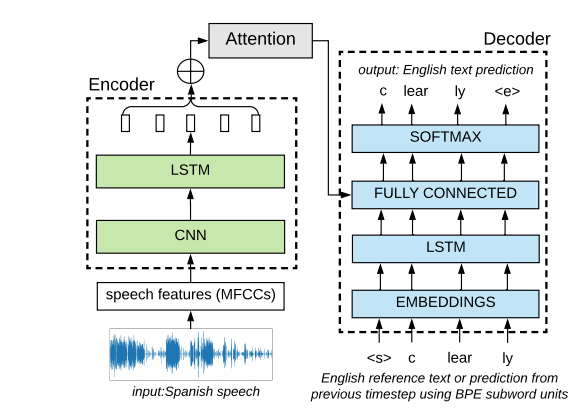
## Review of Literature

Yu and Deng (2015) present Automatic Speech Recognition (ASR) technology as one of the important technologies enabling human-to-human and human-to-computer interactions using speech as the means of communication. While in the past, the lacking speech technology resulted in the field of ASR lagging behind as a means of interaction, with the recent development and findings in the speech technology, ASR has been one of the primaries means of interaction with devices. The continuation of the functioning of Moor’s law and the development of the huge computational power of the processors today using the multi-core processors, Graphical Processing Units (GPUs) has allowed for training of more complex and powerful models which have very high accuracy in terms of speech recognition (Yu and Deng, 2015). Automatic speech recognition also has been seen as the transformation/ conversion of the audio signals or the acoustic structure of the speech signals into its phonemes or implicit phonetic macro structure (Ghai and Singh, 2012). It also can be seen as a speech-to-text conversion engine which converts recognized speech into readable text, which can be further processed.

Current training models, mostly Deep Learning and Neural Network models, require vast amount of labeled audio data for building a decent enough speech recognition engine. The resources also may extend to the phonetic pronunciation dictionaries or lexicons which contains all the words in the language, collection of texts containing more than millions of words. This is a huge amount of work and hence require huge amount of research spanning multiple years to create ASR technology for newer languages. This results in more than 6000 out of 7000 languages being affected since there are not appropriately enough data or resource for the development of ASR technology in these languages as only about 400 languages have over a million of native speakers (Eberhard, Simons and Fennig, 2021). So, building ASR technology for low-resource languages is a tough challenge.

For tackling the problem of low resource for these languages, multilingual resources have been used over the years. Researchers have found that multilingual system outperforms systems trained on single language i.e., monolingual systems (Schultz and Kirchhoff, 2006). The ASR model is first trained in one or more languages, called the source language, and then the model is fine-tuned, trained, and applied to the required language, called the target language. For this process, the implementation of DNNs (Deep Neural Networks) has greatly increased in the recognition performance and accuracy for a variety of tasks (Hinton et al., 2012). One of such implementations is using the Bottleneck (BN) features. Further optimizations on this can be made by identifying the best source languages for the target language allowing for better BN system training and such systems can also be used on non-BN features based systems like hybrid-DNNs (Chuangsuwanich, 2016).

Another way we can tackle the problem for low resource languages is transfer learning. Similar to finding source language for the required target language for pretraining, transfer learning leverages the resources available from a high-resource language for improving the Automatic Speech Recognition performance and accuracy. Pre-training and training of a model for the language using multilingual data has proven to be effective for all of the languages (Toshniwal et al., 2018). Transfer learning has been very effective i.e., pre-training a language model for a high-resource language pair and using the trained parameters for the training of low resource language (Johnson et al., 2017). Inspired by these techniques, Bansal et al. (2019) pre-trained a model in a high resource language, in this case Spanish, and leveraged the trained parameters along with transfer learning and finetuning of the trained parameters for the target language, in this case English language.



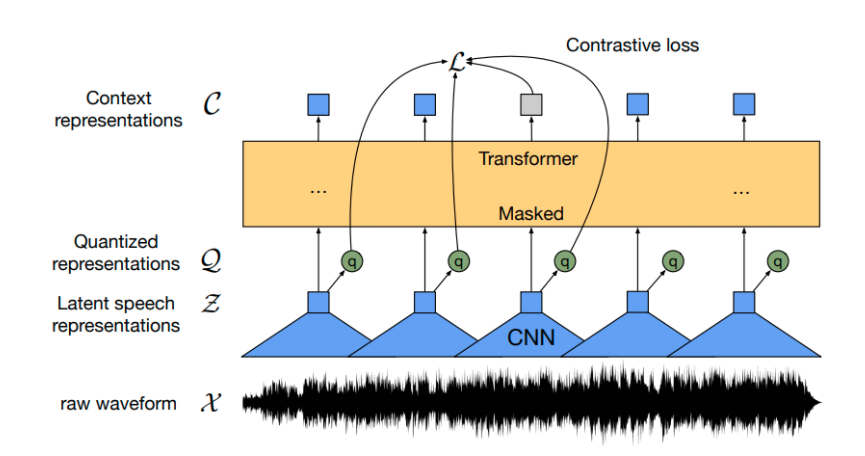
*Fig: Model for ASR and Speech-to-Text using Encoder-decoder with attention model using Spanish word “claro” as pre-training for required English word “clearly”*

Fantaye, Yu et.al. (2020) also used the method of multilingual deep neural network (DNN) modeling methods for the development of a speech recognizer in Chaha language, a low resource language with its own phonological, morphological, and orthographic feature. Compared to the unilingual models, the accuracy and performance improvement was more than 300% allowing for the development of a more effective Chaha ASR system.

Dalmia et.al. (2018) also researched and wrote about the usage of sequence-based multi-lingual and cross-lingual speech recognition for the improvement in performance and accuracy in low resource language scenarios. This would allow for bootstrapping ASR systems allowing for further analysis of new languages limited by low resource and different domains. They show the usage of a Connectionist Temporal Classification (CTC) loss-based end-to-end multi-lingual training of sequence models which is proven effective and shows an overall 6% improvement in terms of WER (Word Error Rate) when compared to the mono-lingual systems.

The project uses the Wav2Vec2 pretrained checkpoint for the development of the Nepali ASR model. For the implementation of Wav2Vec2 in the python project, Hugging Face library has been installed in the project.

Wav2Vec2, released to the public by Facebook AI team of Laexei Baevski, Michael Auli and Alex Conneau on September 2020, is a pretrained model for the development of ASR (Automatic Speech Recognition) technology. Wav2Vec2 is pre-trained on the raw representations of audio data from more than 50,000 hours of unlabeled speech using the novel contrastive pretraining objectives. By masking feature vectors randomly on the speech representations before passing these vectors into the transformer network, the model learns about the contextualized speech representations which helps in making predictions as required (Platen, 2021)



*Fig: XLSR approach using feature encoder and masking (Conneau et al., 2020)*

As shown in the figure, the model is trained through the multilingual quantized speech units that are produced through the feature encoder representations. The output embeddings are used for training the Transformer model through contrastive learning. This approach is very similar to BERT’s masked language modeling.

The development of this model has allowed for the creation of a state-of-the-art ASR system which with very little actual labeled/ transcribed audio data can achieve the results or even perform better than the current models used. For the English data set of LibriSpeech, using only 10 minutes of labeled data, Wav2Vec2 had an error rate of less than 5%.

# Review of Technologies

### Software

#### Python

Python is a high-level general-purpose programming language. The availability of built-in data structures with dynamic binding helps make it a popular choice for RAD (Rapid Application Development) which also is often times used as connecting language for different built components. The usage of Python is simple and easy due to its syntax which emphasizes readability and also contains a multitude of library and packages, which helps in code reusability. Also, the Python interpreter is freely available with most of the standard library having open-source license. The most current version of Python is 3.9.5 but due to model and package version requirements, the project uses v3.7.5.

#### Jupyter Notebook

Jupyter Notebook is an open source webapp that allows the user for creating and editing a JSON-based documents that contains live computer code (eg. Python) along with texts and visualizations as required. Project Jupyter is the owner and maintainer of Jupyter Notebook. It is based off the IPython project and ships with a default IPython kernel allowing for programming in Python. It also supports more than 100 other kernels which can be used. The notebook documents are human readable with analysis and results along with executable code for data analysis. The app is based on server-client architecture allowing the execution of notebook code through the browser. The execution can be run on the local desktop or on the server through SSH.

#### Jupyter Lab

Jupyter Lab is a web-based UI application for Project Jupyter. JupyterLab allows the user to work on different documents like Jupyter notebooks, terminals, consoles, and other components. It also can help act as an Integrated Development Environment for developing applications. Another feature of JupyterLab is the parsing, editing, and viewing of the different file formats like images, CSV, PDF and so on. This helps to develop the whole project in one environment rather than swapping between different environments for different file types.

# Methodology and Design

## Research setting

The research is mainly focused on the low resource language. Due to the familiarity with the Nepali language, Nepali language with Devanagari script has been chosen as the pilot language for the study. So, the study is primarily focused on the native Nepali language speakers of Nepal. Since Nepali language does not have a lot or research invested into Automatic Speech Recognition, Speech-to-Text technologies, Text-to-Speech technologies, or Natural Language Processing, it does not have a lot of accurate labeled data. While there are a lot of unlabeled data which can easily be obtained through scraping; transcribing and labeling these audios can be expensive and requires time. Hence, there exists a lack of proper resource for proper and accurate enough ASR solutions using current most commonly used ASR algorithms like DeepSpeech since they require a huge amount of data and according to estimates needs almost a year for planning to the final development stage. This causes a lot of enthusiastic developers to stray away from this problem and hence stopping any progress made in this domain.

Same as Nepali language, there are around 6000+ languages around the world that face the same problem of lack of enough resources for training an accurate enough ASR solution.

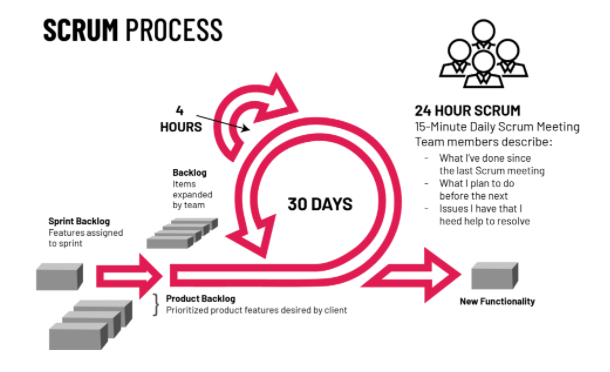
## Research process

For the development of the final product for the project, Agile Scrum has been used as the methodology to follow in software development.

Agile software development is the software development methodology which primarily centers around the iteration of product development. The idea of iterative development is the main focus where the requirement and solution changes and evolves through production development and change as required in those production build. The Agile development greatly enables the quicker development and output from teams with higher degree of adaptability to change.

Scrum is an Agile methodology where product development is split into Sprints, a timeframe for the completion of a set amount of work by teams or individuals. The Agile Scrum helps in the coping of changing requirements allowing for better estimates and better control of the project schedule and state. As opposed to the waterfall model, where all the product requirements are defined beforehand, in Agile Scrum, as the product development goes on newer requirements may be added as per the client need or the availability of timeframe.

For the project, the Sprint length was 1 week as there was need to go through multiple iterations quickly for reaching the product requirement and goals; hence the need for multiple changes as the product was being developed.



*Fig: Scrum Process*

While waterfall methodology was considered as an alternative solution for the project, waterfall methodology fails to work with changing needs of the project. The project had an initial goal of developing an ASR engine with required metrics for evaluation, but the development of project could not be fully planned as all the information about the process was not known during the start of the project. While the project was ongoing, there was a need for the methodology to adapt to the changing needs of the output for proper optimization of the ASR model along with the changing requirement of the engine. Hence, the Agile Scrum process was chosen as the better alternative among the two.

## Data Collection

For the process of data collection i.e., the labeled audio data in Nepali language, research was the primary means of collection. For research, the Internet was used. Available datasets freely available to use or with free educational license usage were searched through the usage of search engines. Popular dataset hosting websites like Kaggle, Mozilla Common Voice, LibriSet, OpenSLR and other such sites were searched through for the availability of appropriate dataset i.e., ASR dataset for Nepali language. One of such datasets was available at OpenSLR.org with the link http://openslr.org/54/ . The dataset contained a total of more than 157,000 utterances in Nepali language for ASR training with the identifier of SLR54. This provided us with a set of transcribed audio data for Nepali language with appropriate wave files (audio files) and a TSV file containing the transcript of the audio files along with the mapped audio filename (Kjartansson et al., 2018). This was used as the primary training dataset.

Wav2Vec 2.0 model was used for the training of the model with the base model pretrained on 16kHz sampled audio. The base model for fine-tuning was collected from Facebook’s XLSR-Wav2Vec2. The model was pre-trained on 56K hours of speech in 53 different languages. Then, using the previous OpenSLR 54 for fine-tuning, we trained our final Nepali language ASR model.

For the evaluation metrics of the model, another dataset from OpenSLR was used, SLR43. This dataset contains multi-speaker TTS data for the Nepali language, codenamed ne-NP. The data set contains 2064 high-quality transcribed audio data for the Nepali language. The dataset contains a set of wave files and the map of the transcript, and the filename is present in a TSV file (Sodimana et al., 2018).

## Data Analysis

The audio data files downloaded from OpenSLR both SLR54 and SLR43 were manually verified for quality checks and corruption checks. Since there were a large amount of data, the files were cross validated with randomly chosen files from among the datasets.  
Also, the TSV files were checked with corresponding mapped audio files to check if the transcription is correct as required.

Pandas library and datasets module of Hugging Face was used for data import and analysis. All the transcripts were checked for the presence of unwanted special characters and were filtered out as necessary, some transcripts were removed from the dataset while in some, only the special characters were removed from the transcript.

Also, due to GPU memory limitations in the training machine, audio files greater than 6 seconds (recommended length by Hugging Face’s Transformers doc) were filtered out from the dataset for training and testing purposes. For this, librosa library of python was used.

## Limitations

The research approach for data collection has its own limit. While there existed the required dataset for Nepali languages, it is not hundred percent sure that other low-resource language would have freely licensed dataset available online. Also, even with the extensive research done on the Internet, only the OpenSLR 54 and OpenSLR 43 dataset were found online. While these data are enough for initial training and give a good enough performance and accuracy value, in the long run for improving the ASR engine performance, the same method of data collection cannot be used.

For tackling this problem, we can contact the usage of LSPs (Language Service Providers) who have datasets available for training or who can label/ transcribe the provided audios as required. They also can help in collecting audios for provided list of transcripts.

# Implementation and Testing

The implementation of the project is the product development i.e., Automatic Speech Recognition for Nepali language. As defined in the project aims, the output of the ASR engine must be in Devanagari script. The implementation of the project also depends on the fulfilment of project requirements, listed in the requirements using MoSCoW rule in project plan. Along with the requirements predefined in the project plan, any changes and modification required for the better product development must be taken into consideration and should be applied/ used for the new Sprint planning since Agile Scrum development methodology has been used for product development.

## Dataset/ Training data acquisition

The dataset was downloaded from the OpenSLR website individually using the Google Chrome downloaded in .zip format. All the files were extracted to a single folder. The audio files were cross validated for quality checking and corruption check. Some of the audio files were chosen randomly and were opened and through the use of the website MetaData2Go (<https://www.metadata2go.com/>), the metadata of the audio files along with their quality were manually verified. The TSV file mapping the transcript to the audio files were also cross validated with random files/ rows of data being chosen and the checking the corresponding audio files if the transcript is correct.



*Fig: Sample data of the OpenSLR 54 dataset*

## Data Preprocessing

### Data Import

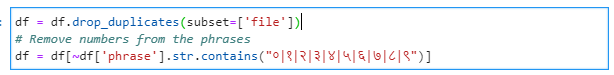
For the purpose of data pre-processing, a Jupyter Notebook was created. Pandas’ library was used for importing the TSV file into Python using the function read\_csv which converted the imported TSV file into a DataFrame for further processing. The file name was prepended with required path variables for correctly referencing the required files from the Python script.



*Fig: CSV importing and path pre-pending*

### Data Cleaning

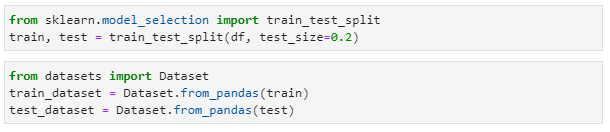
The duplicate data in the DataFrame were removed with the checking subset being the filename, if present using the DataFrame method, drop\_duplicates. Using a regex, all the phrases using Devanagari numbers "०|१|२|३|४|५|६|७|८|९" were removed from the dataframe.



*Fig: Removal of duplicates and phrases with Devanagari script numbers*

### Test/Train split

Using sklearn module (from scikit-learn), the whole dataset was split into training and testing split with the ratio of 8:2 i.e., 80% for training and 20% for testing. Also, the split datasets were converted to Dataset object from Hugging Face Transformers for further processing.

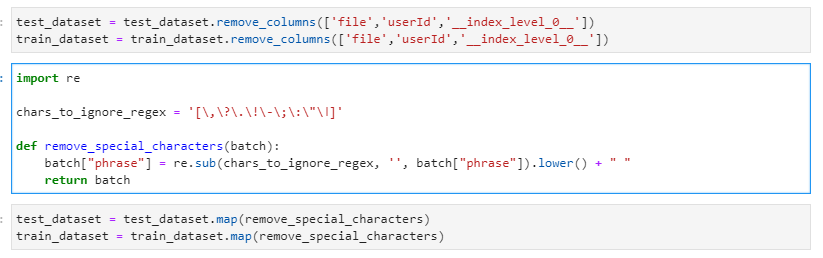


*Fig: Train and test split and DataFrame to Dataset conversion*

### Further Data cleaning

Unnecessary data were cleaned from the dataset using remove\_columns method of Dataset.

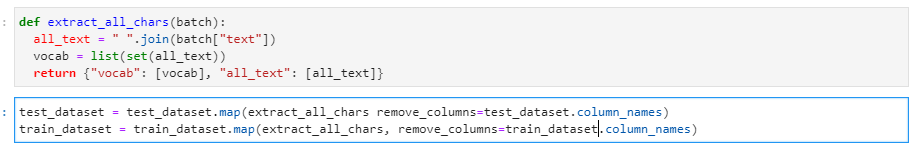
Also, the transcription list contained special characters such as **,?.!-;:”|**. Since without a corresponding language model for parsing these special characters, these characters hold no meaning as they hold no phonetic details for training. Also, for parsing the audio signal to transcript, these special characters are not required for training. Hence, characters that were deemed unnecessary for the training of the data were removed from the dataset using the map method of Dataset.



*Fig: Further data cleaning*

### Vocabulary Creation

For Connectionist Temporal Classification (CTC), it is required to convert the speech chunks into letters. So, we extract the distinct characters/ letters from the input dataset (both train and test dataset) and build a vocabulary consisting of these characters. For this purpose, a mapping function is created which converts the list of transcripts to set of chars.



*Fig: Character Extraction map function*

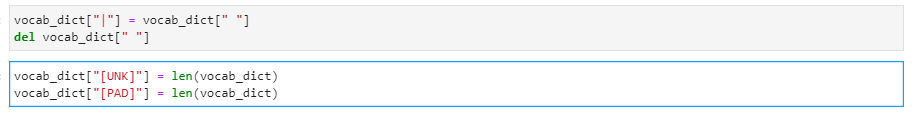
Then, a union of the distinct characters of the train and test dataset is created and is converted to an enumerated dictionary. Also, we see that the dataset contains some unnecessary characters in the transcription. We will remove these characters from the vocabulary.



*Fig: Vocabulary creation and vocabulary data cleaning*

For CTC, the “ “ i.e., a blank string corresponding to a blank speech or split in words, we convert it to a more visible character. Also, we create an unknown token “UNK” for allowing the model to handle unknown speech and characters not included in the training set.

Along with that, a padding token corresponding to CTC’s blank token “PAD” is created. The “blank token” is considered to be a key component for the training of CTC algorithm.



*Fig: Required token creation*

The vocabulary is then exported into a json file which we use later for Wav2Vec2 CTC.

### Audio Loading and Resampling

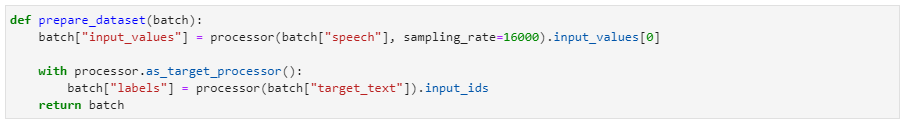
We use torchaudio module for loading the audio files into speech array for processing. Also, since our base model XLSR-53 is trained on 16kHz audio files, we need to convert the speech array into 16kHz. For this, we use Resample function of the torchaudio.tranforms module. For the above purpose, we create a mapping function.



*Fig: Map function for audio loading and resampling*

### Final Dataset Preparation

Now, for the last step of data preprocessing, we convert the dataset to the format required by the model for training. For this, we create a mapping function which extracts the “input\_values” from the loaded audio files which included normalized data. Then, the required transcription is encoded to label ids for training process.



*Fig: Data preparation*

Finally, the dataset is saved into disk such that the reuse of the dataset is easy and does not require to re-process the whole dataset again.

## Training

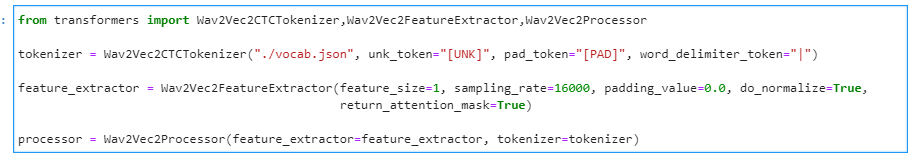
The dataset from the data preprocessing is loaded from the disk.

### Tokenizer, Feature Extractor and Processor Creation

Using Hugging Face’s Transformers, we create a Wav2Vec2CTC tokenizer object using the created vocabulary.

Also, a feature extractor is created using the same Transformers module with required parameters.

Using the instantiated tokenizer and feature extractor, a Wav2Vec2 processor is created.



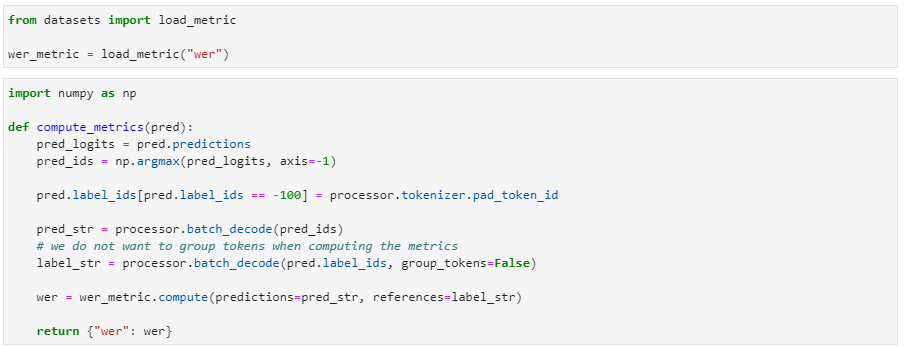
*Fig:*  *Tokenizer, Feature Extractor and Processor Creation*

### Data Collator

In our case, XLSR models have much larger input length than the output length. Some of the input sizes vary hence we create padding for the training samples to the longest sample which is more efficient. For this, we create a padding data collator.

### Evaluation Metric

As defined in the project requirements, WER (Word Error Rate) is the primary evaluation metric. For this purpose, we will use load\_metric function of datasets module to load the WER metric for further prediction. Then, a map function is created for the calculation of WER.



*Fig: Map function for metrics computation*

### Base Model import

We then import the pretrained model facebook/wav2vec2-large-xlsr-53 for training. Also, since XLSR model consists of CNN layers stack as the first component which has acoustic meaning but no contextual meaning for the features from the raw speech signal which has already been sufficiently trained during pretraining, they are not needed to be further trained.



*Fig: Model importing*

### Training arguments and trainer

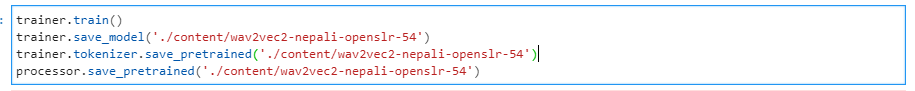
We then set the required training arguments fit for the model training. Then, using the created instances, we create a Trainer object for training our model.



*Fig: Training arguments and Trainer*

### Training Start

Finally, the model can be started to be trained. The training of the model took around 54hrs, but this can be significantly decreases using additional GPU computational power as required in the future.   
After training completion, the model is saved into the required folder.



*Fig: Training start and model exporting*

## API Creation

The API for accessing the creating was created by creating a class file Nepali\_ASR. For allowing users to access the model without Python setup, a client-server side architecture was preferred since it would allow for the users to easily access the model API without any other setup required.  
The server development was still done on Python using the Flask model. An endpoint was created for allowing the users to upload the file for transcription. Since parsing long text at once is difficult due to hardware limits, files longer than 30s are split into chunks and parsed separately. After all the chunks have been parsed, the transcriptions are then joined and then finally returned.

Also, another API endpoint was created to allow the users to access the API through the use of file URLs hosted publicly instead of uploading the files.

## Front-end Development

The front-end of the product was created to allow users to access the API through GUI usage. The webpage was developed using simple HTML/CSS/JS. The webpage allows the user to record audio and get the transcript for the audio.

The website also allows for the user to upload audio files as required and get the transcript for the audio file.

## Product Testing

Testing of a product is vital to ensure the quality and performance of the product as required. It helps to detect any bugs that may occur in the program or code breaking errors or memory/ CPU usage management issues that may arise after the release of the product. Since the product consists of multiple stages i.e., actual ASR engine exposed through class APIs, Flask-based server API for accessing the ASR engine through clients and the GUI/Front end portion of the product. For all the product stages, Black Box testing was used for carrying out the testing of the product. The usage of black box allows the tester to emulate the end-user, who may or may not have programming knowledge.

Postman was used for testing the API endpoints.

### Testing of the ASR engine

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Test Requirement** | **Result** |
| 1. | ASR engine accepts multiple formats of audio as input (WAV, MP3, FLAC) | PASS |
| 2. | ASR engine can accept audio URL as input | PASS |
| 3. | ASR engine can handle audio length greater than 30 seconds | PASS |
| 4. | ASR engine accepts audio file and return transcript as a JSON object | PASS |
| 5. | The output transcript is in Devanagari Script | PASS |

### Testing of the API

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Test Requirement** | **Result** |
| 1. | API is successfully hosted and accessible to the user | PASS |
| 2. | API successfully connects with the ASR engine | PASS |
| 3. | API has multiple endpoints, for input as audio file and input as URL | PASS |
| 4. | Both the endpoints return the transcript in JSON format | PASS |

### Testing of the Front-end (Webapp)

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Test Requirement** | **Result** |
| 1. | User can record audio | PASS |
| 2. | User can upload audio files | PASS |
| 3. | User can insert URL for transcription | PASS |
| 4. | User can view the transcript | PASS |

# Product Evaluation

The major aim of the product was the creation of an Automatic Speech Recognition (ASR) engine that can convert speech/ audio data in Nepali language to text in Devanagari script. The whole product was based on the principle of parsing of audio data through waveforms and phonetics usage for creating the transcript in Nepali/ Devanagari script through the usage of Wav2Vec 2.0 algorithm. Throughout the different stages of the product development from the research phase, different methods were used for tackling the issues that popped up. The whole project was research based along with the data collection being from the Internet.

Going through the whole process again, for data collection, audio data from actual people with different accents would have been preferred along with the current data collected. But this process is thoroughly time consuming as compared to the current method. The development process i.e., training of the whole model; coding, evaluation, and testing were done on the go. So, there were many times the code structure and flow had to be changed for the betterment of the project output with better accuracy (WER) and performance values. This also would have saved a lot of development time for the model training. For this step, sufficient research on existing works should have been done during this whole process. Also, help from experts on the field of Automatic Speech Recognition (ASR) or supervisors should have been taken which would also had saved a lot of time for the whole process.

The output of the product i.e., the ASR engine, had a better-than-the-rest accuracy with a Word Error Rate (WER) of 4% on the evaluation dataset of OpenSLR-43. The WER was lower than the expected WER of 10%. Hence, we can conclude that the ASR engine performed excellently.

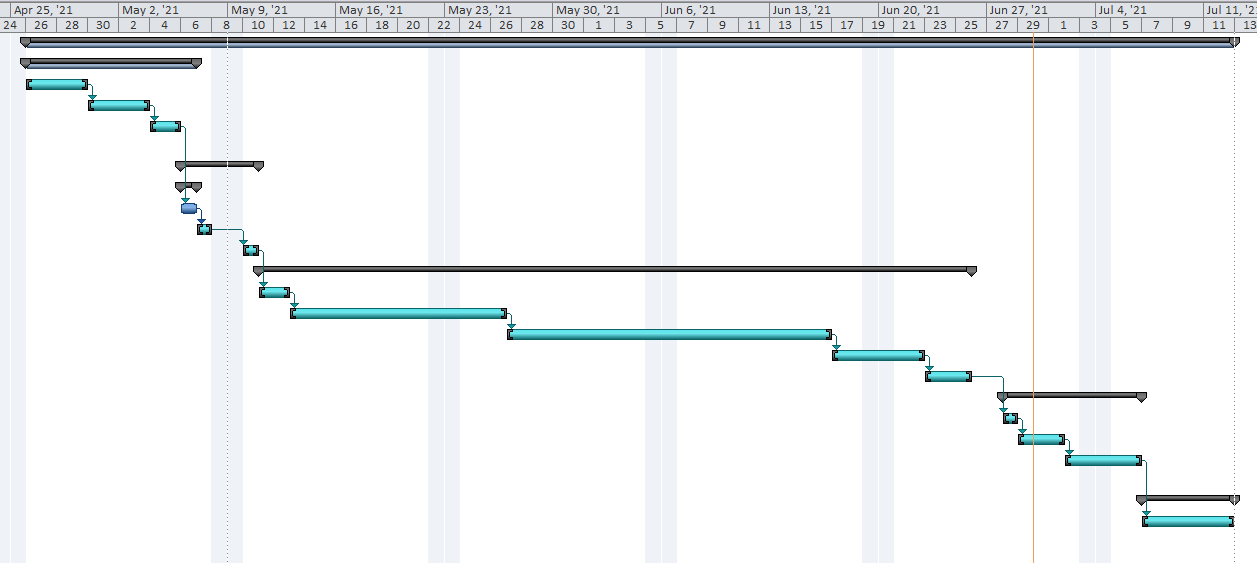
Also, APIs were created successfully using Python OOP and Flask which allows the usage of the ASR engine through a client-server architecture. The model generated can also easily be fine-tuned using the same code where the pre-trained would be the current model generated.

The training of the model was heavily dependent on a powerful computing machine. In this case, my personal PC was used. While it is a decent enough system for machine learning, most of the time was wasted while waiting for the training to complete which took an approximate time of 52 hours to train on the 157,000 utterances. The model was trained for only 5 epochs. So, with an adequately powerful machine, this training time could be brought down significantly along with increasing the training epoch which would increase the accuracy of the ASR engine.

# Project Management and Evaluation

## Project Management

### Gantt Chart



### Work Breakdown Structure

## Project Evaluation

The project was an accumulation of Machine Learning, Deep Learning, Client/ Server architecture as well as API based applications with the end goal of creating Nepali language ASR engine with the output in the Devanagari script format. The aim of the project was the development of speech to text technology for Nepali language which was successfully developed during the project duration.

# Summary and Conclusion

## Summary

The end goal of the research was the creation of an Automatic Speech Recognition engine/ model for low-resource languages where accumulating a large amount of labelled data is difficult, time and money consuming tasks. For the pilot of the ASR engine, Nepali language was chosen with the output being formatted in Devanagari script. The accomplishment of this research would allow people to have a boilerplate for training a speech-to-text engine for even their own local language. The product, i.e., Nepali ASR engine was successfully trained with low amount of labelled audio data and the results were very satisfying for data used for the training. The data collection for the whole process was completely research based where all the labelled data were collected through the Internet from open-source repositories, namely OpenSLR 54 for training and OpenSLR 43 for evaluation. The training of the project output i.e., the ASR engine was successfully done within the expected parameters and input required i.e., under 1000 hours of labelled audio data. Also, the evaluation of the model was within expectations of 10% WER.

## Conclusion

The project focus of development of an ASR product/ engine for low resource language is important for human-computer interaction since most of the 7000 languages spoken in the world do not have enough speakers. But since audio is the prime form of communication for humans, the usage of speech to text plays a vital role to have these languages support interaction with the computers in easier form. The project would also pave way for further processing of the audio form of data, for example, Natural Language Processing, Sentiment analysis and so on. The possibilities are endless with this. The developed ASR engine has been trained for Nepali language but can easily be changed/ migrated to train languages other than Nepali with few steps. Almost all of the training code and parameters have already been integrated and hence can easily be used with parameters optimizations, as necessary.

## Future works

The algorithm used in the project, Wav2Vec 2.0, is still in its early phase, so as time goes on, changes and optimization to the usage of the algorithm would help in increasing the accuracy of the model along with fewer data as required for the training of the model. For the product, it has been noted that timestamping the output of the product is needed and can help in multiple other sectors regarding the usage of the ASR engine. So, automatically generated timestamps for the transcripts are one of the future works. While some work has been done regarding timestamps, it is not fully tested and functional hence, not production ready. Usually, most of the ASR engines like AWS Transcribe, Google Speech-To-Text use technology to generate a confidence score which tell how confident the engine is in the transcribed words. These scores can help in choosing the right transcript/ words from the transcript and as we go forward, words and phonemes with low confidentiality scores can be trained using fine-tuning of the models which will help in increasing the accuracy of the product.

Output of the current project, the development of an ASR product would allow for various opportunities to be linked with voice capabilities. For example, in the case of Nepali language, multiple works are ongoing regarding the Natural Language Processing using texts i.e., Devanagari scripts. If a viable ASR product is made available, this would allow NLP to be done and processed on the audio form of Nepali data. Also, this would pave the way for the development of Voice Assistants and AIs in these languages.

# Bibliography

* Singh, M., 2016. Automatic Speech Recognition System: A Survey Report. *Science & Technology Journal*, 4(2), pp.152-155.
* Baevski, A., Conneau, A. and Auli, M., 2020. *Wav2vec 2.0: Learning the structure of speech from raw audio*. [online] Facebook AI. Available at: <https://ai.facebook.com/blog/wav2vec-20-learning-the-structure-of-speech-from-raw-audio/> [Accessed 13 June 2021].
* Baevski, A., Zhou, H., Mohamed, A. and Auli, M., 2020. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. *arXiv:2006.11477 [cs.CL]*.
* Americaslibrary.gov. 2021. *The First Telephone Call*. [online] Available at: <https://www.americaslibrary.gov/jb/recon/jb\_recon\_telephone\_1.html> [Accessed 18 June 2021].
* Manjutha M., Gracy J., Subashini P. and Krishnaveni M., 2021. Automated Speech Recognition System – A Literature Review. *International Journal of Engineering Trends and Applications (IJETA)*, [online] 4(2). Available at: <http://www.ijetajournal.org/volume-4/issue-2/IJETA-V4I2P9.pdf> [Accessed 18 June 2021].
* Lam, K., 2021. *The History of Automatic Speech Recognition*. [online] Deepgram. Available at: <https://deepgram.com/blog/the-history-of-automatic-speech-recognition/> [Accessed 18 June 2021].
* O’Shaughnessy, D., 2008. Invited paper: Automatic speech recognition: History, methods, and challenges. *Pattern Recognition*, 41(10), pp.2965-2979.
* Schneider, S., Baevski, A., Collobert, R. and Auli, M., 2020. Wav2Vec: Unsupervised Pre-training for Speech Recognition. *arXiv:1904.05862*, [online] 4. Available at: <https://arxiv.org/pdf/1904.05862.pdf> [Accessed 18 June 2021].
* Yu, D. and Deng, L., 2015. *Automatic Speech Recognition*. 1st ed. Springer.
* Ghai, W. and Singh, N., 2012. Literature Review on Automatic Speech Recognition. *International Journal of Computer Applications (0975 – 8887)*, [online] 41(8). Available at: <https://research.ijcaonline.org/volume41/number8/pxc3877646.pdf> [Accessed 19 June 2021].
* Eberhard, D., Simons, G. and Fennig, C., 2021. *Ethnologue:Languages of the World*. 24th ed. Dallas, Texas: SIL International.
* Schultz, T. and Kirchhoff, K., 2006. *Multilingual Speech Processing*. Burlington: Elsevier.
* Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. and Kingsbury, B., 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6), pp.82-97.
* Chuangsuwanich, E., 2016. Multilingual Techniques for low resource automatic speech recognition. [online] Available at: <https://apps.dtic.mil/sti/pdfs/AD1040167.pdf> [Accessed 19 June 2021].
* Toshniwal, S., Kannan, A., Chiu, C., Wu, Y., Sainath, T. and Livescu, K., 2018. A Comparison of Techniques for Language Model Integration in Encoder-Decoder Speech Recognition. *arXiv e-prints*, [online] pp.arXiv:1807.10857. Available at: <https://ui.adsabs.harvard.edu/abs/2018arXiv180710857T> [Accessed 19 June 2021].
* Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda B. Viegas, Martin Wattenberg, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google’s multilingual neural machine translation system: Enabling zero-shot translation. Trans. ACL, 5:339–351.
* Bansal, S., Kamper, H., Livescu, K., Lopez, A. and Goldwater, S., 2019. Pre-training on high-resource speech recognition improves low-resource speech-to-text translation. *arXiv 1809.01431*, [online] Available at: <https://arxiv.org/abs/1809.01431> [Accessed 19 June 2021].
* Dalmia, S., Sanabria, R., Metze, F. and Black, A., 2018. Sequence-Based Multi-Lingual Low Resource Speech Recognition. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*,.
* Kjartansson, O., Sarin, S., Pipatsrisawat, K., Jansche, M. and Ha, L., 2018. Crowd-Sourced Speech Corpora for Javanese, Sundanese, Sinhala, Nepali, and Bangladeshi Bengali. *The 6th Intl. Workshop on Spoken Language Technologies for Under-Resourced Languages*.
* Sodimana, K., De Silva, P., Sarin, S., Kjartansson, O., Jansche, M., Pipatsrisawat, K. and Ha, L., 2018. A Step-by-Step Process for Building TTS Voices Using Open Source Data and Frameworks for Bangla, Javanese, Khmer, Nepali, Sinhala, and Sundanese. *The 6th Intl. Workshop on Spoken Language Technologies for Under-Resourced Languages*.
* Platen, P., 2021. *Fine-Tune Wav2Vec2 for English ASR in Hugging Face with 🤗 Transformers*. [online] Huggingface.co. Available at: <https://huggingface.co/blog/fine-tune-wav2vec2-english> [Accessed 20 June 2021].
* Conneau, A., Baevski, A., Collobert, R., Mohamed, A. and Auli, M., 2020. Unsupervised Cross-lingual Representation Learning for Speech Recognition. *arXiv 2006.13979*, [online] Available at: <https://arxiv.org/abs/2006.13979> [Accessed 20 June 2021].