COPYRIGHT

ACKNOWLEDGEMENT

We'd like to start off by conveying deep and heartfelt respect to the Institute of Engineering, TU for maintaining the academic environment where students like us are able to experiment and create interesting projects to put their knowledge into practice. We further want to express our sincere gratitude to the Department of Electronics and Computer Engineering, Pashchimanchal Campus for providing us with the opportunity to work on this minor project as a part of our course work.

We would like to thank our project supervisor Hari Parsad Baral sir for his continuous support and guidance throughout the project. We would also like to acknowledge the help and guidance of our other teachers without which we couldn't have finished this project.

We would like to thank our friends and colleagues for their reviews and suggestions during the development of the project. We will also like to express profound appreciation to our parents and family members for their constant support. They have directly or indirectly helped us get to this level where we're able to apply our knowledge in pursuit of progress.

ABSTRACT

Dhwoni is an automatic speech recognition system that uses an RNN-CTC, LSTM model. We show that an end-to-end deep learning approach can be used to recognize Nepali speech. We trained the model with about twenty thousand 5-second audio samples from openslr.com/54 and achieved a consistent Character Error Rate of 29.5% on the test dataset. We experimented with various model architectures and algorithms to tune our system to achieve the lowest CTC loss. The experiments were performed using the freely available Google Colab facility. The model we designed takes an approximate training time of 40 minutes in Google Colab (with GPU Acceleration). Further, we also designed a crowdsourcing system to help us curate speech samples from speakers all over Nepal. We developed a web app for crowdsourcing speech samples (www.dhwoni.com) where users can record audio samples for predetermined text. They can also flag previously recorded samples as correct or incorrect. Based on the collected data, we curate a moderate-quality crowdsourced dataset.

TABLE OF CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

CTC: Connectionist Temporal Classification

RNN: Recurrent Neural Network

CNN: Convolutional Neural Network

LSTM: Long Short-Term Memorization

ASR: Automatic Speech Recognition

NaN: Not a Number

MFCC: Mel Frequency Cepstral Coefficient

GMM: Gaussian mixture models

HMM: Hidden Markov Model

CUDA: Compute Unified Device Architecture

TSV: Tab Separated Values

OpenSLR: Open Speech and Language Resources

ZWNJ: Zero-Width Non-Joiner

ZWJ: Zero-Width Joiner

LIST OF TABLE

LIST OF FIGURE

1. INTRODUCTION

1.1 Background

Language is what makes us human. It is the most natural form of expression. Speech in particular is an extremely powerful form of communication. Recent advancements in Artificial Intelligence technology have made it possible for us to no longer rely on contrived actions like mice, touch, keyboard etc to interact with computers. Technology has become powerful enough to understand user intent and behave accordingly.

Dhwoni approaches to model an ASR system for the Nepali language by using Deep Speech 2 as its base model and also incorporating a number of recent advancements developed to improve the performance of ASR systems made using the RNN-CTC model. The entire research effort surrounding the project is targeted to explore the effects of these minor changes.

One major obstacle in making an ASR system for the Nepali Language is the unavailability of voice data sets. Which is why we decided also to develop an application for the purpose of collection and curation of an extensive data set for the Nepali language (which as of today is very limited, the only publicly available data set being the one made available by Google in OpenSLR) [1]. The data set consists of audio files, and a TSV file of the corresponding transcriptions.

The unavailability of training resources at our disposal forced us to use only the resources made freely available by Google Colab and Kaggle to run our experiments and tune our model. Resource bottlenecks have had a major impact on our project, so much so that we had to resort to training only a small portion of the dataset. Furthermore, since the systems provided free of cost are highly unstable, we could not train our models in parallel pipelines. More than 100 experiments were run, each only as long as about 40 minutes, because given the limited resources, training the 10GB dataset was not possible.

The long term goal of this research and development effort is to take a substantial and significant first step towards the eventual reality where technology is accessible to the average Nepali who doesn't have a good command in the English language. Currently, language is a massive barrier to a ton of modern technologies like the internet, and a significant portion of Nepali population hasn't been able to reap the benefits of modernization. The step we are taking with Dhwoni is to make a decent open-source Automatic Speech Recognition software. The end result we aim to achieve is a tool which will convert spoken Nepali voice into Nepali text. In later steps, we aim to integrate this ASR with Natural Language Processing systems, and make it available to all willing developers via a Cloud API so that developers can make their software accessible to everyone. We are also collecting crowdsourced labeled speech dataset which we will make available in a free license so other developers can use it in their ASR systems and other research efforts.

1.2 Problem Statement

Much work has been done in development of ASR systems for various languages. On the other hand, the languages of Nepal used to barely receive any attention. A number of state-of-the-art ASR systems, for the Nepali language, have been developed recently. However, research about how these systems achieved their state-of-the-art functionality have not been made publicly available. It should also be noted that the services of one such state-of-the-art ASR system, made by Google, is available exclusively to the users of Google Chrome and Google Keyboard. The unavailability of extensive voice datasets of the Nepali language also adds up to this void. Which is why exploration in this domain is an endeavor full of pitfalls and a rocky one at that.

1.3 Objective

The objectives for the project are as follows:

1. Develop and deploy an application for dataset collection.

   Major features of the application are as follows:

   - o Allow users to add new speech data by reading out the displayed text.
   - o Allow users to validate the existing data, recorded previously in the application, by listening and then validating the recording by comparing it with the corresponding textual transcription.
2. Design a model for speech recognition system and train it using the datasets.
3. Deploy the ASR system in the cloud as a web app.

1.4 Scope

Listed below are the various probable applications of Dhwoni ASR:

- As an intuitive mode of input for interacting with technology.
- As a simple communication medium for Illiterate and Disabled people.
- For building convenient virtual assistants on modern mobile devices.
- For transcribing spoken text from videos to help people hard of hearing.

2. LITERATURE REVIEW

2.1 Related Works

Limited research has been done in the field of continuous, speaker-independent speech recognition in Nepal, partly due to limited datasets in the public domain. Speech recognition is a difficult task in itself because of the contextual relations of words and sentences, causing misinterpretations of what the speaker means to say or convey.

Audio data primarily contains environment noises and other information of the human voice such as gender, emotion, age et cetera. Feature extraction of human voice data allows for the best parametric representation of acoustic signals. The non-parametric method for modeling human auditory perception systems, Mel Frequency Cepstral Coefficient (MFCC) in particular, does so by assuming the known variations of the human ears' perceivable critical bandwidth. MFCC is a set of coefficients derived from the cepstral representation of the audio recording. The cepstrum (which is the inverse Fourier transform of the logarithm of the estimated signal spectrum) is mapped onto the Mel scale, which estimates the human auditory estimation of the relative perceived frequency differential into a non-linear scale. [2].

Previous generation ASR models used Gaussian mixture models (GMM)-HMM but recent advancements in computing have made it possible to use neural networks. Neural network nodes, that are independent in nature, were previously computed sequentially. The development of Compute Unified Device Architecture (CUDA) enabled parallel computation by harnessing the power of GPUs, thus speeding up the computation of gradients. Convolutional Neural Networks (CNNs) in particular make it possible to model spatial and temporal relations in audio datasets, thus allowing the extraction of relevant features. [3]

Speech recognition requires memorization of long-term sequential relations. Vanilla RNN retains the temporal relations with its recurrent units but a vanishing gradient problem arises in case of long-term dependencies. LSTM allows long-term memorization by gating its update, enabling learning of longer sequences and storing of previously occurred samples. [4]

In speech recognition, every input audio data frame corresponds to an output transcript. Direct alignment of input to output is not appropriate for speech recognition because of the variational and independent nature of people's way of speaking. Connectionist Temporal Classification (CTC) assigns probability of an output Y (transcript) to an input X (RNN output) and computes the loss between the input and output sequences. [5]

The Deep Speech 2 model is a highly extensive model for speech recognition that uses RNN-CTC. Deep Speech 2 uses a huge dataset and also uses complex feature extraction methods that includes acoustic models, language differences, pronunciation models, speaker adaptations, etc. The model attempts to make a single engine able to learn to be as competent as human beings and also be able to generalize a variety of languages/applications with only minor modifications. It also attempts to learn new languages from scratch without having to make dramatic changes. [6]

A more recent model proposes an entirely different architecture to learn long-range dependencies. The Transformer model uses self-attention layers (Scaled dot product attention and Multi-head attention) that learns long-range dependencies without recurrent layers. Layer outputs can be calculated parallelly, thus reducing computation complexities and making RNNs seemingly obsolete. [vaswani2017attention,] The Transformer-Based Acoustic Modeling for Hybrid Speech Recognition by Facebook AI outperforms state-of-the-art RNN-CTC applications in ASR. This model, when compared to state-of-the-art RNN models on Librispeech datasets and Facebook's internal dataset yields lower word error rates (WERs). [7] The Transformer model is still under active research and shows promising results in sequence modeling tasks.

One such effort in ASR that closely matches Dhwoni was published in the International Journal of Computer Applications, titled Nepali Speech Recognition using RNN-CTC Model, which gives some context regarding the work being done in Nepali ASR in Nepal. However, since the model was trained on a smaller, skewed dataset, the model adapted to the training patterns instead of generalizing the features. [6] Apparently no other conclusive research is publicly available for Nepali ASR till date.

Google's state-of-the-art speech recognition system for the Nepali language is publicly available for application purposes and shows significant performance. However, the architectures and algorithms used have not yet been publicly disclosed.

3. METHODOLOGY

3.1 Overview

There are several technologies and techniques used to perform automatic speech recognition in the world. A high-level explanation is given below:

The dataset from OpenSLR is a clean dataset with a low variance. In our project, we take the audio data from the first two zip files of the OpenSLR dataset, rejecting any samples greater than 5 seconds. The sound samples lower than 5 seconds are padded with 0’s at the end. In the next step, MFCC is calculated for the sound waves and 81 features are extracted. Now, the Amplitude-Time sound signal has turned into a MFCC Spectrogram.

We’ve selected 71 characters (including a blank character) in the Nepali dataset.
['क', 'ख', 'ग', 'घ','ङ','च', 'छ', 'ज', 'झ', 'ञ', 'ट', 'ठ', 'ड', 'ढ', 'ण', 'त', 'थ', 'द', 'ध', 'न', 'प', 'फ', 'ब', 'भ', 'म', 'य', 'र', 'ल', 'व', 'श', 'ष', 'स', 'ह', 'क्ष', 'त्र', 'ज्ञ', 'अ', 'आ', 'इ', 'ई', 'उ', 'ऊ', 'ए', 'ऐ', 'ओ', 'औ', 'ा', 'ि', 'ी', 'ु', 'ू', 'े', 'ै', 'ो', 'ौ', 'ं', 'ँ', '्', 'ृ', 'ः', '०', '१', '२', '३', '४', '५', '६', '७', '८', '९', ' ']

The spectrogram is then fed into a CNN1D layer to extract temporal features from the dataset. Furthermore, in the CNN layer, Padding is done to keep the shape of the dataset. We’ve used ten kernels of CNN1D having the stride of 2. The layer is then fed into a dropout layer with a dropout value of 0.1. The layer is further normalized.

After that, there is a Dense Layer unit composed of two Linear-Dense layers. Each

Linear-Dense has the following elements ordered sequentially.

- Linear

- LayerNorm
- GELU
- Dropout

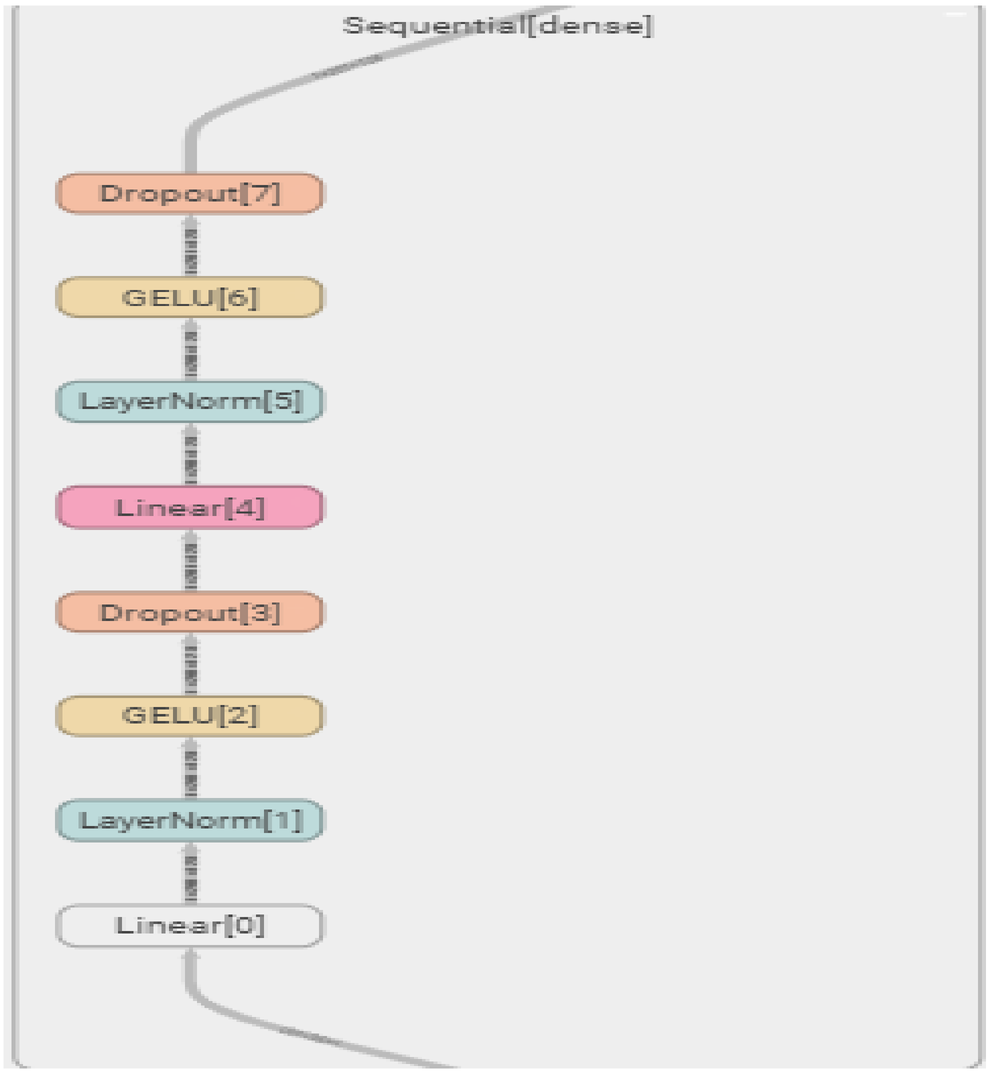


Figure 3.1: Linear fully connected layer unit

Second layer in the process is the Recurrent Layer. Recurrent Layer is an LSTM layer purely based on deep learning literature. Experimenting with different RNN units like GRU, stacked LSTM etc. was not feasible with our lack of availability of resources. The LSTM layer has an input size of 128 and a hidden size of 1024.

After coming from the LSTM layer, the outputs are then fed to a Dense layer with softmax activation and the CTC loss is calculated, which is further back propagated.
AdamW optimizer is used to optimize the weights and biases of the network. To tackle the problem of exploding gradients, Gradient Clipping is done. OneCycleLearning Rate Scheduler was used to schedule the learning rate as it provides faster convergence.

3.2 Implementation Details

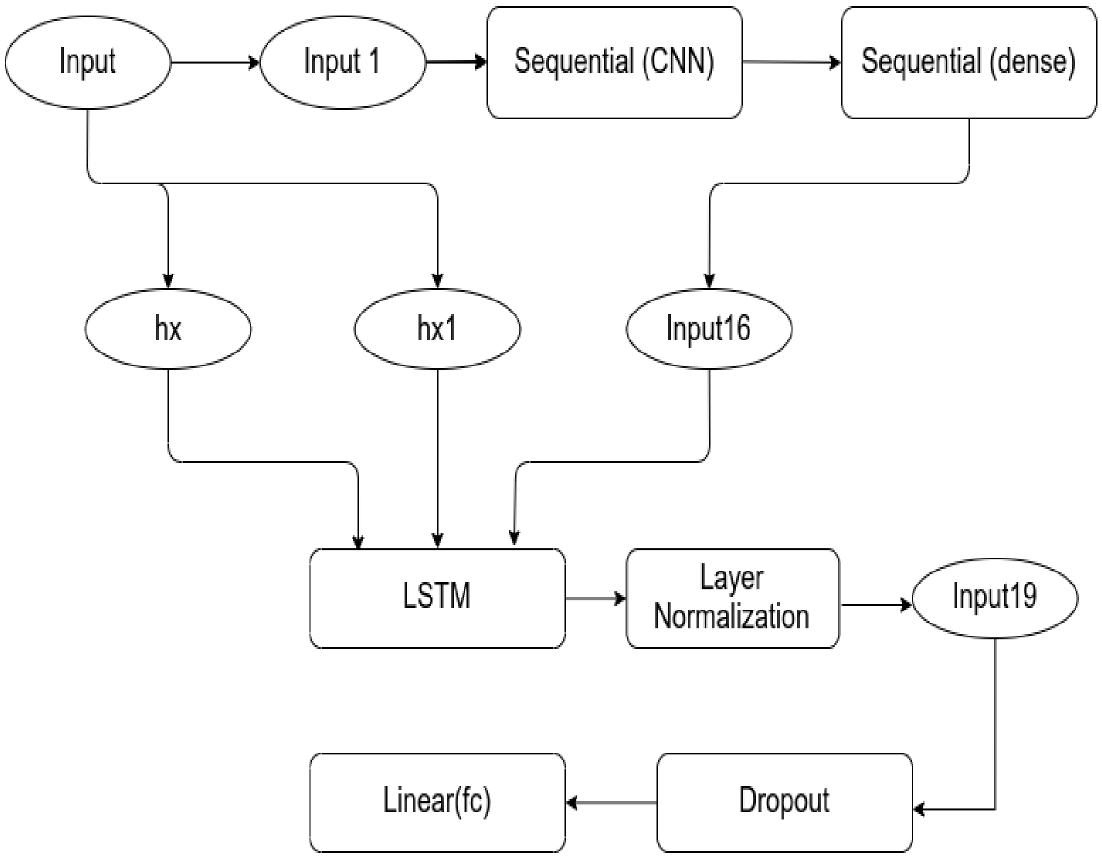


Figure 3.2: High level model architecture

Our system takes MFCC as the basis for all further processing. It is fed into a sequential network consisting of the combination of a Convolutional Neural Network and a dense layer. Each unit has its own normalization step and appropriate dropouts are applied at each unit. The output of this step is fed to LSTM which is also normalized and dropout applied. Finally, we feed it to a Linear fully connected network and calculate CTC loss which is back-propagated.

3.2.1 MFCC

Sound is a wave. To store sound in a computer, waves of air pressure are converted into voltage via microphone. It is then sampled with an analog-to-digital converter where the output is stored as a one dimensional array of numbers. Simply put, an audio file is just an array of amplitudes sampled with a certain rate known as sampling rate. As metadata an audio file has sample rate, number of channels of sound and precision (bit depth).

A standard telephone audio has a sampling rate of 8 kHz and 16-bit precision.
The term bit-rate (bit per second) which is sometimes used to measure the overall quality of an audio.

$$bitrate = samplerate * precision * noofchannels$$

A raw audio signal is high-dimensional and difficult to model as there is much information but consists of many unwanted signals especially in case of human speech sound.

Mel-frequency Cepstrum Coefficients (MFCC) tries to model the audio in a format where it performs those types of filtering that correlates to human auditory systems and their low dimensionality. It is the most commonly used feature for Automatic Speech Recognition (ASR). Mel-frequency Cepstrum (MFC) is a representation of the short-term power-spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear Mel-scale of frequency and MFCC are the coefficients that collectively make up the MFC.
Following are the steps to compute MFCC.

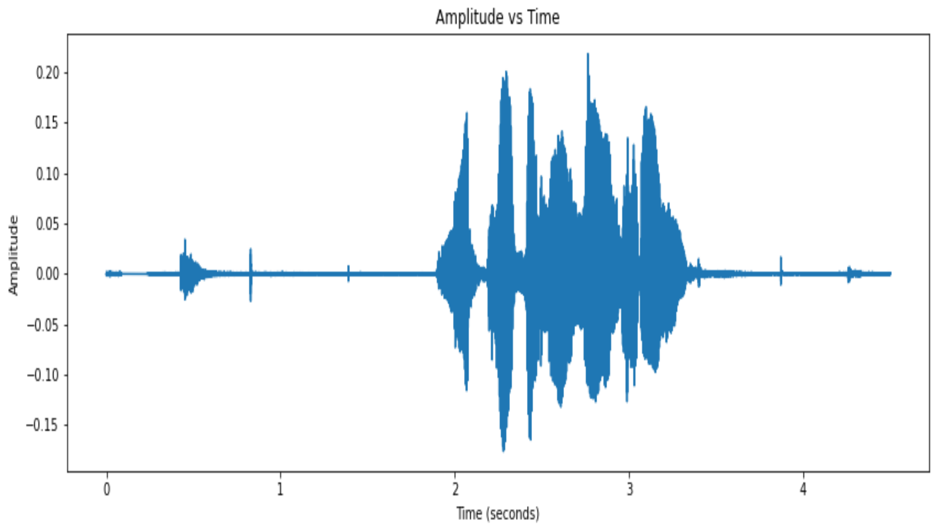


Figure 3.3: Amplitude vs time graph of sound wave

3.2.1.1 Pre-emphasis

This is the first step in feature generation. In speech production, high frequencies usually have smaller magnitudes compared to lower frequencies. So in order to counter the effect we apply a pre-emphasis signal to amplify the amplitude of high frequencies and lower the amplitude of lower frequencies.

If x(t) is the signal, $y(t) = x(t) - \alpha x(t-1)$ where, α is generally 0.95 or 0.97.

Here's the visualization of a pre-emphasized signal.

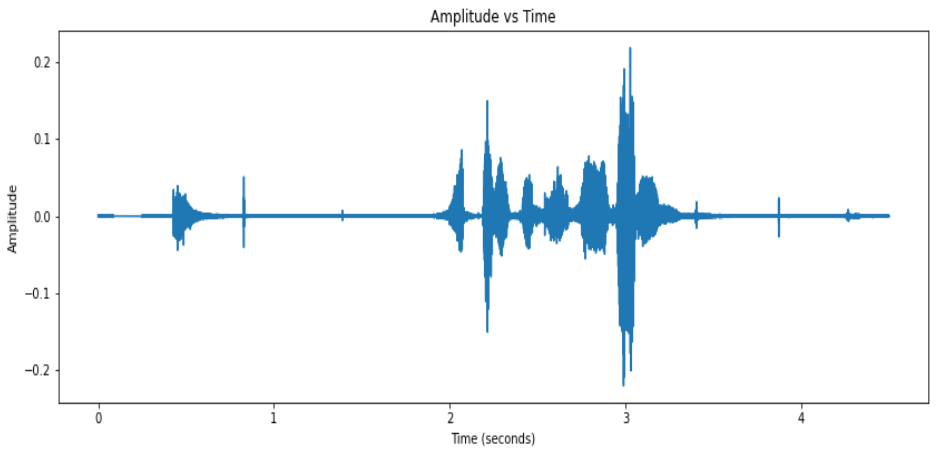


Figure 3.4: Sound wave visualization after pre-emphasis

3.2.1.2 Framing

Acoustic signals are perpetually changing in speech. But studies show that the characteristics of voice signals remain fixed in a short interval of time (called quasi-stationary signals). So while modelling the signal we take small segments from the audio for further processing.

Separating the samples into fixed length segments is known as framing or frame blocking. These frames are usually from 5 milliseconds to 100 milliseconds. But in case of speech signal to preserve the phonemes, we often take the length of 20-40 milliseconds, which is usually the length of phonemes with 10-15 milliseconds overlap.

These segments are later converted to frequency domain with an FFT.

Why do we use overlapping of the frames?

We can imagine a non-overlapping rectangular frame. Each sample is, somehow, treated with the same weight. However, when processing features extracted from two consecutive frames, the change of property between the frames may induce a discontinuity, or a jump ("the difference of parameter values of neighboring frames can be higher"). This blocking effect can create disturbance in the signal.

3.2.1.3 Windowing

Windowing multiplies the samples by a scaling function. This is done to eliminate discontinuities at the edges of the frames. If the function of windows is defined as w(n), $0 < n < N-1$ where N is the number of samples per frame, the resulting signal will be;

$s(n) = x(n) * w(n)$

Generally hamming windows are used where

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right)$$

Why do we use windowing?

When we are changing signals from the time domain using FFT, we cannot perform computations on infinite data points with computers so all signals are cut off at either end. For example, let's say we want to do FFT on pure sine waves. In the frequency domain, we expect a sharp spike on the respective frequency of the sine wave. But when we visualize it, we see a ripple-like graph on many frequencies which is not even the frequency of the wave.

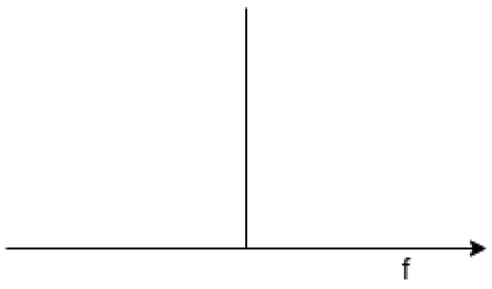


Figure 3.5: Graph of the expected outcome

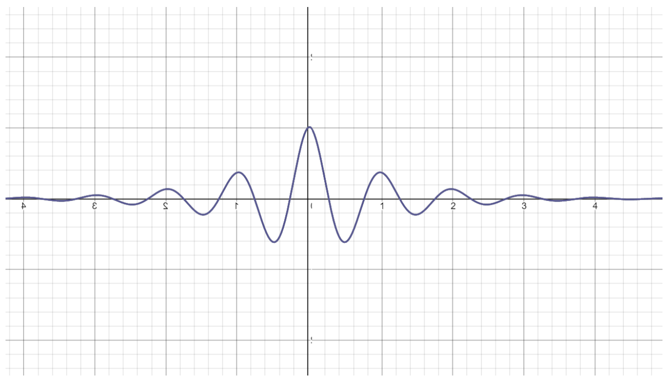


Figure 3.6: Graph of the Observed Outcome

When we cut off signals at either end, we are indirectly multiplying our signal by a square window. So there are a variety of window functions people have come up with. Hamming window, analytically, is known to optimize the characteristics needed for speech processing.

3.2.1.4 Fourier Transform

Fourier transform (FT) is a mathematical transform that decomposes a function (often a function of time, or a signal) into its constituent frequencies. This is used to analyze frequencies contained in the speech signal. And it also gives the magnitude of each frequency.

$$X(k) = \sum_{n=0}^{N-1} x(n).e^{-\frac{2\pi ikn}{N}} \quad \text{where } k = 0,\ 1,\ 2\ldots\ N - 1$$

Short-time Fourier transform (STFT) converts the 1-dimensional signal from the time domain into the frequency domain by using the frames and applying a discrete Fourier transform to each frame. We can now do an N-point FFT on each frame to calculate the frequency spectrum, which is also called Short-Time Fourier-Transform (STFT). STFT provides the time-localized frequency information because in speech signals, frequency components vary over time. Usually we take N=256.

Spectrogram: A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. For some end-to-end systems, spectrograms are taken as input. This helps in 3D visualization of the FFT.

Magnitude of Spectrogram

$$S_m = |FFT(x_i)|^2$$

Power Spectrogram

$$S_p = \frac{S_m}{N}$$

Where N is the number of points considered for FFT computation. (typically 256 or 512)

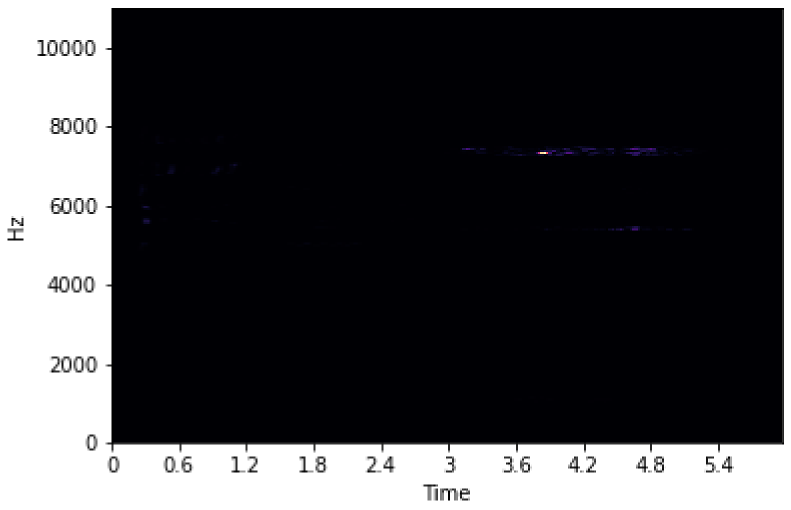


Figure 3.7: FFT spectrogram

3.2.1.5 Mel-Filter Bank

The magnitude spectrum is warped according to the Mel scale in order to adapt the frequency resolution to the non-linear properties of the human ear by being more discriminative at lower frequencies and less discriminative at higher frequencies. We can convert between Hz scale and Mel scale using the following equations:

$$Mel(f) = 2595 * log(1 + \frac{f}{700})$$

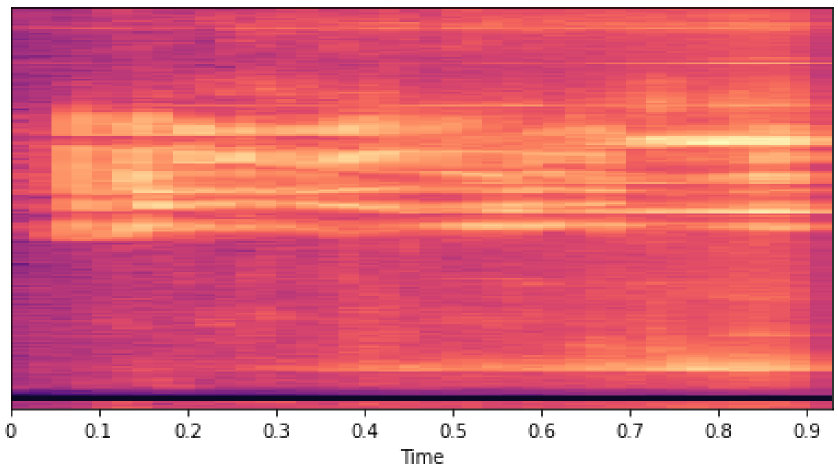


Figure 3.8: MFCC spectrogram

3.2.2 Convolution Neural Network

A CNN works for identifying patterns and extracting features from data which can then be further used to form complex patterns within higher layers. A 1D CNN is efficient when needed to derive features from fixed-length segments. CNN 1D has been shown to perform better on sequential data. As CNN 1D only has to sum a one-dimensional array of features, as opposed to CNN 2D which has to operate on a quadratic set of features it has a time complexity of O(n). Thus, it produces much faster.

Forward and Backward Propagation in CNN1D layers:

In CNN 1D, the forward propagation is:

$$x_k^l = b_k^l + \sum_{i=1}^{N_{l-1}} conv1D(w_{ik}^{l-1}, s_i^{l-1})$$

Where,

x_k^l = Input

b_k^l = bias of k^{th} neuron at layer l

S_i^{l-1} = output of the i^{th} neuron at layer l-1

w_{ik}^{l-1} = Kernel from the i^{th} neuron at layer l-1 to the k^{th} neuron at layer l

Conv1D(..) is used to perform the convolution operation

The intermediate output, y_k^l can be expressed by passing the input x_k^l through activation function $f(\cdot)$ as:

$y_k^l = f(x_k^l)$ and ↓ $s_k^l = y_k^l ss$

Where, s_k^l stands for the output of the $k^t h$ neuron of the layer l and "ss" represents the downsampling operation with a scalar factor ↓ss.

3.2.3 Long Short Term Memory (LSTM)

Humans don’t start thinking from scratch every second. We understand each word based on the understanding of the previous words. We create context from the events in our mind before processing the current word being spoken. But the traditional neural networks cannot do this and that’s the major shortcoming of traditional neural networks. For example, let's suppose that we want to know what kind of event is happening in any novel. It’s unclear how traditional NNs do it because it cannot reason about the previous event before analyzing what is happening in the present time.

RNN addresses this issue by implementing cells where the context of the previous data is stored so it can be helpful to define knowledge in future data.

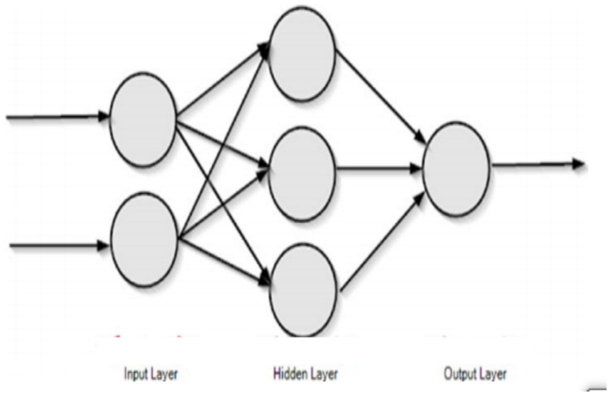


Figure 3.9: Traditional neural network

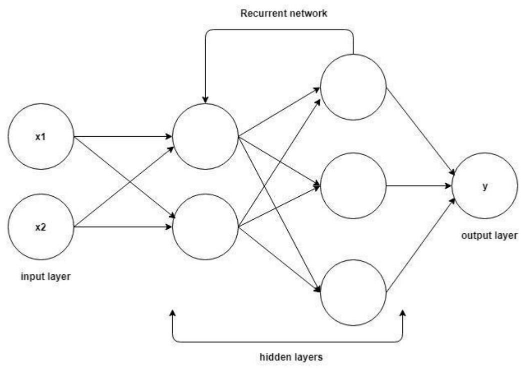


Figure 3.10: Recurrent neural network

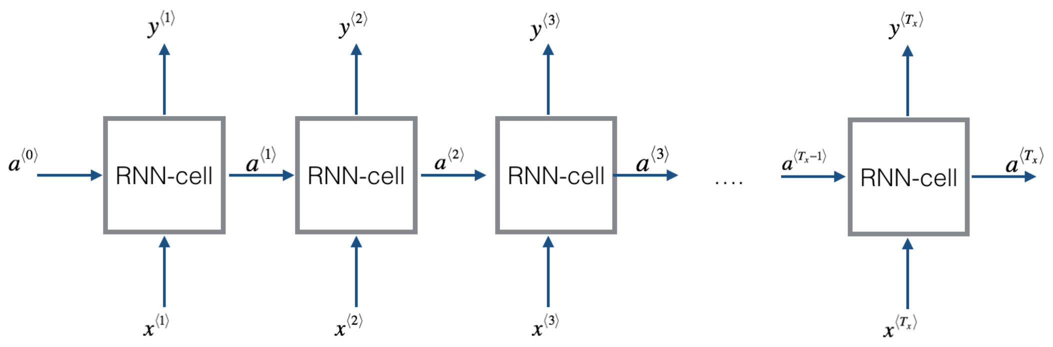
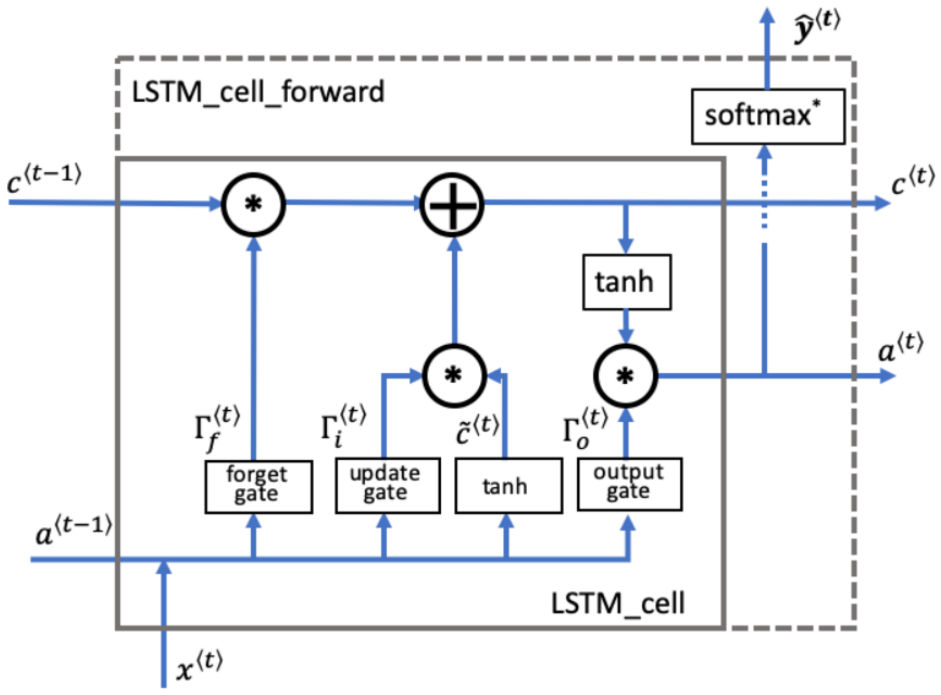


Figure 3.11: RNN Architecture

LSTM is a recurrent neural network architecture used in deep learning. Unlike feedforward neural networks, LSTM has feedback connections. An LSTM unit consists of a cell, an input gate, an output gate and a forget gate. The cell remembers the value over arbitrary time intervals and the three gates are used to regulate the information flow in the cell.



$\hat{y}^{\langle t \rangle}$
LSTM_cell_forward
softmax*
$c^{\langle t-1 \rangle}$
$c^{\langle t \rangle}$
tanh
$a^{\langle t \rangle}$
$\Gamma_f^{\langle t \rangle}$
$\Gamma_i^{\langle t \rangle}$
$\tilde{c}^{\langle t \rangle}$
$\Gamma_o^{\langle t \rangle}$
forget gate
update gate
tanh
output gate
$a^{\langle t-1 \rangle}$
LSTM_cell
$x^{\langle t \rangle}$

Figure 3.12: LSTM unit architecture

3.2.3.1 LSTM Cell:

It acts as a memory unit which stores the context data of past input in the time sequence data.

$$c^{<t>} = \Gamma_i \times c^{\sim<t>} + \Gamma_f \times c^{<t-1>}$$

The current cell value depends on the candidate memory cell update and previous cell value. All the gate values provide the control for the update of the cell value.

3.2.3.2 Input Gate:

The input gate is also known as the update gate and this gate decides what aspect of candidate cell value ($c^{\sim<t>}$) to add to the current cell state ($c^{<t>}$).

$$\Gamma_i^{<t>} = \sigma(W_i[a^{<t-1>}, x^{<t>}] + b_i)$$

When $\Gamma_i^{<t>}$ is close to 1, it allows the present candidate cell value to be passed to the hidden state.

When $\Gamma_i^{<t>}$ is close to 0, it prevents the present candidate to be passed into the hidden state determining that the current input is not important for storing to the memory cell.

3.2.3.3 Forget Gate:

The forget gate decides what aspect of previous cell value ($c^{<t-1>}$) to add to the current cell state ($c^{<t>}$).

$$\Gamma_f^{<t>} = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

When $\Gamma_f^{<t>}$ is close to 0, the LSTM unit will forget the state of the previous cell.

When $\Gamma_f^{<t>}$ is close to 1, the LSTM unit will mostly remember the state of the previous cell.

Output Gate ($\Gamma_o^{<t>}$)

The output gate decides what gets sent as the prediction of the time step.

$$\Gamma_o^{<t>} = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

It depends upon the previous hidden state a<t-1> and current input x<t>.
By using the output gate, the hidden state is determined and hence the name output gate.

$$a^{<t>} = \Gamma_o^{<t>} * tanh(c^{<t>})$$

3.2.3.4 Candidate Cell Value

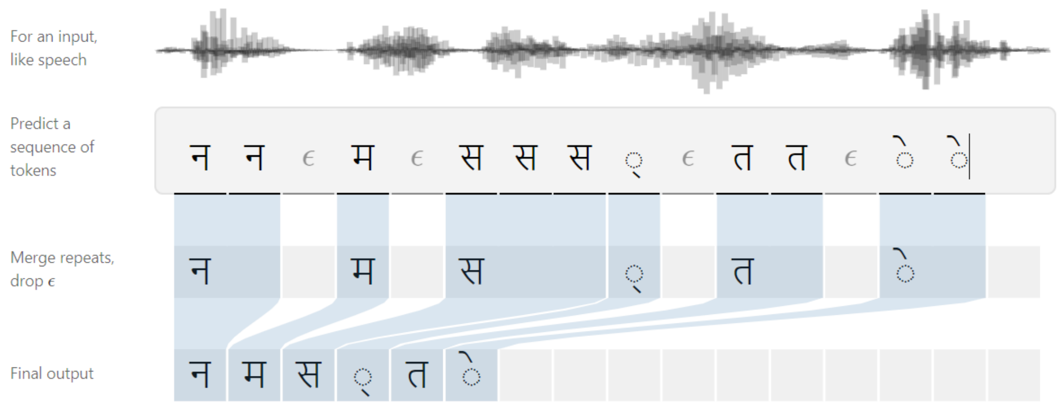
The candidate cell value is the information of current input which gives the context of the present state. The candidate cell value $c^{\sim<t>}$ may or may not be the final cell state value depending upon the value of input and forget gate.

$$c^{\sim<t>} = tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

3.2.4 CTC Loss

CTC is a way to solve the problem that we don't know the alignment between input and the output. It is used in many-to-many sequence problems.

For example,



For an input, like speech
Predict a sequence of tokens
न न ϵ म ϵ स स स ़ ϵ त त ϵ े े
Merge repeats, drop ϵ
न म स ़ त े
Final output
न म स ़ त े

Figure 3.13: CTC loss visualization

Here in the dataset, the location of the exact output of the wave is not defined and we have to deal with it automatically where the CTC decoder shines.

Formally,

Let's consider input sequences $X = \{x_1, x_2, ... x_T\}$ such as audio to corresponding output sequences $Y = \{y_1, y_2 ... x_U\}$ such as transcripts.

And the problem is that we want to find an accurate mapping of x to y.

The simpler supervised learning may lead to following problems

- The input and output sequences may vary
- The ratio of length of of x and y may vary
- The accurate alignment of x and y is not defined

The CTC algorithm overcomes these challenges by giving the output distribution over all possible Ys from X and we use this distribution either to infer a likely output or access the probability of given output.

Mathematically,
Let L be the set of labels and L' be the set of labels with the blank label.
For a sequence of length T, we denote the set of possible paths $L'^{T} = \pi$
Given the sequence of inputs x and labelling z, where $|z| \leq |x|$, we try to maximize the probability of the labelling given the sequences (maximum likelihood estimation).

$$\hat{\theta} = argmax_{\theta} \prod_{i=1}^{N} P(z^{(i)} | x^{(i)};\ \theta) \text{ .... } (*)$$

Let's now define a many-to-many map B: $L'^{T} \rightarrow L^{\leq T}$ that maps the length T label sequence of characters L' to their labelling equivalent in L, while removing all the blanks and repeated labels. In effect, B performs the collapsing operation.
For example,

B(_क__ललल_मम) = B(क_ल_मम_) = कलम

Now we define B^{-1} to map a label sequence z to the set of all possible label sequences (paths in $\boldsymbol{\pi}$) that collapse to z. So,

$$\{B(x) \mid x \in B^{-1}(y)\} = y$$

Now, we can consider the likelihood of a labelling z as the sum of probabilities of all the paths that can collapse to z.

$$P(z \mid x; \theta) = \sum_{\pi \in B^{-1} z} P(\pi \mid x;\ \theta)$$

After substitution, the equation (*) becomes;

$$\hat{\theta} = argmax_{\theta} \prod_{i=1}^{N} \sum_{\pi \in B^{-1} z^{(i)}} P(\pi \mid x^{<i>};\ \theta)$$

Now changing the problem from maximization to minimization and converting the product of sum to sum of product, the equation becomes,

$$\hat{\theta} = argmin_{\theta} \sum_{i=1}^{N} log[\sum_{\pi \in B^{-1} z^{(i)}} P(\pi \mid x^{<i>};\ \theta)]$$

3.2.4.1 Variants of CTC Loss

CTC actually describes a family of loss functions. The formulation with blanks is just the original example. Other examples:

• If we can guarantee that the same label cannot occur twice in succession, we can create a version of CTC without blanks.

• There are times when there are multiple types of blanks, perhaps one for each possible non-blank symbol.

• Can even follow certain finite state machines.

For our loss function we used CTC Loss with $blank = 72$.

3.2.5 Gaussian Error Linear Unit (GELU) Activation

An activation function ensures faster and better convergence of neural networks. GELU works by combining properties from dropout, zoneout and RELUs. A relu and dropout both yield a neuron's output with the relu deterministically multiplying the input by zero or one and dropout stochastically multiplying inputs by zero. The RNN regularization technique called zoneout stochastically multiplies inputs by one.

In GELU, we merge all those functionalities by multiplying input by zero or one stochastically while also depending upon the input. And GELU performs well with language processing and speech tasks. [8]

Mathematically,

$$x * m \sim Bernoulli(\phi(x))$$

Where, $\phi(x) = P(X \leq x)$, $X \sim N(0, 1)$

which is the cumulative distribution function of standard normal distribution.

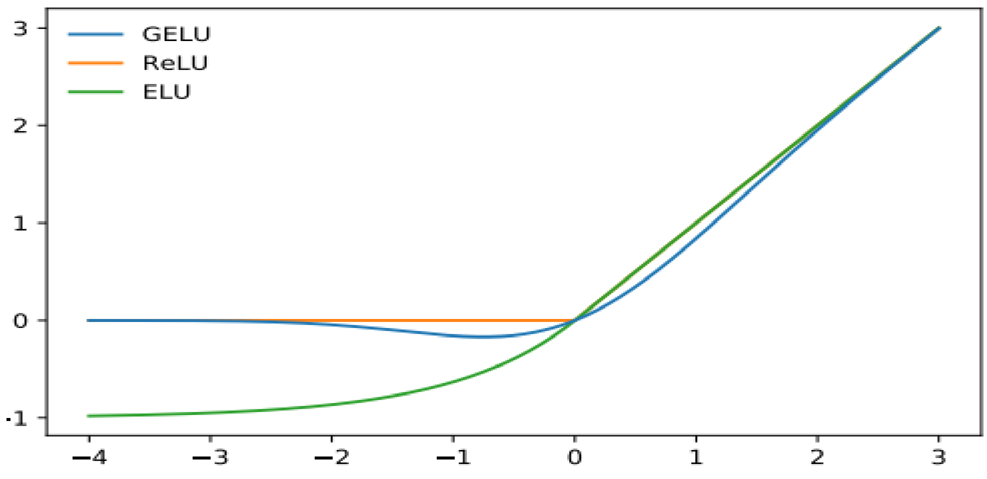


Figure 3.14: GELU vs ReLU vs ELU

Here;

$GELU(x) = x\,P(X \leq x) = x * \phi(x)$,

which can be approximated as,

$$GELU(x) = 0.5x * [1 + tanh(\sqrt{2/\pi} * x + 0.044715x^3)]$$
$$= x * \sigma(1.702)\backslash$$

3.2.6 Layer Normalization

Layer Normalization directly estimates the normalization statistics from the summed inputs to the neurons within a hidden layer so the normalization does not introduce any new dependencies between training cases.

It works well for RNNs and imposes both training time and generalization performance of several existing RNN models.

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2}$$

Where H is the number of hidden units in the layer l.

3.2.7 AdamW

Adaptive Moment Estimation (Adam) computes adaptive learning rates for each parameter. In addition to storing and exponentially decaying average of past squared gradients (v_t), Adam also keeps an exponentially decaying average of gradients m_t, similar to momentum.

$$m_t = \beta m_{t-1} + (1 - \beta) g_t$$

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^{2}$$

Where, m_t and v_t are estimates of the first moment (the mean) and the second moment (uncentered variance) of the gradients respectively.

As m_t and v_t are initialized with 0's, they are biased towards zero.

To counteract these biases, the first and second moment estimates are computed as:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

Then, update rule for adam is:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+e}\hat{m}_t$$

Despite superior training time, Adam in some cases does not converge to an optimal solution, so for some tasks, best results are achieved only by applying SGD with momentum.

L_2 regularization and Weight Decay:
Networks with smaller weights are observed to overfit loss and generalize better. The rate of weight decay per step w defines the relative importance of minimizing the original loss function and finding small weights.

Weight Decay:

$$\theta_t = (1 - \lambda)\theta_{t-1} - \alpha\nabla f_t(\theta_{t-1})$$

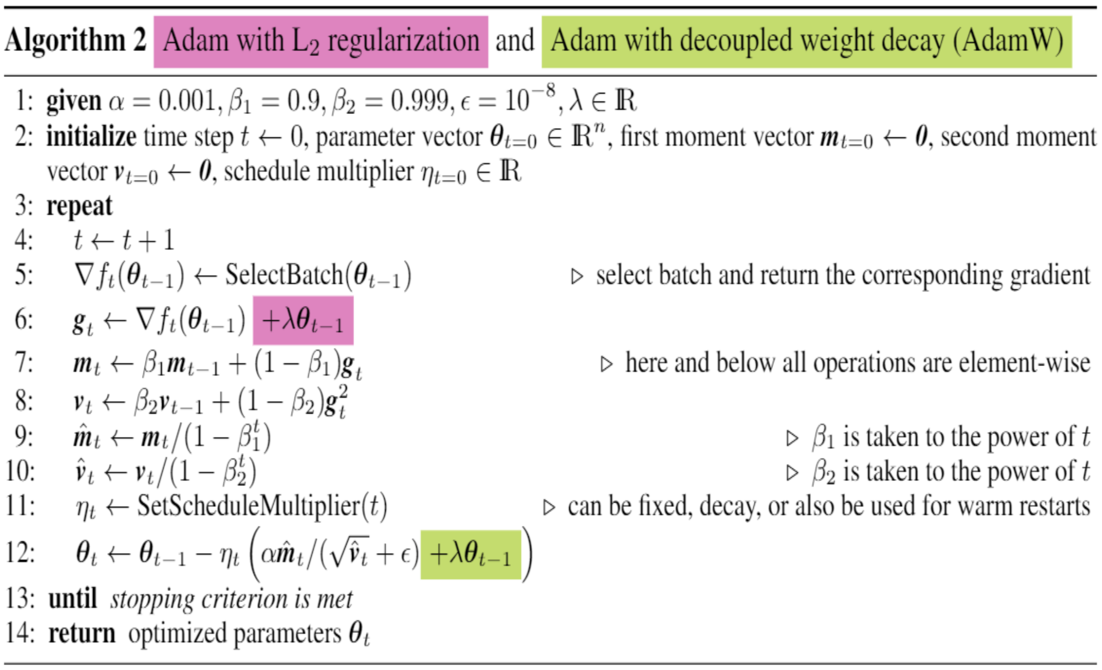
Where λdefines the rate of weight decay per step

$\nabla f_t(\theta_t)$is the batch gradient to be multiplied by a learning rate α.

We'll notice the additional term $-\lambda\theta_{t-1}$that exponentially decays the weights θ and thus forces the network to learn smaller weights.
Often, instead of performing weight decay, a regularized loss function is defined (L_2 regularization).

$$f_t^{reg}(\boldsymbol{\theta}) = f_t(\boldsymbol{\theta}) + \frac{\lambda'}{2}\|\boldsymbol{\theta}\|_2^2, \text{ with } \lambda' = \frac{\lambda}{\alpha}$$

As Weight decay equals L_2 regularization for standard SGD, they are often used interchangeably, including in popular deep learning libraries. However as demonstrated by [9], this equivalence doesn't hold for adaptive gradient methods. So, a new modification is done to recover the original formulation of weight decay regularization by decoupling the weight decay from the optimization steps taken with respect to the loss function. This decouples the optimal choice of weight decay factor from the setting of the learning rate for both standard SGD and Adam and also substantially improves Adam's generalization performance. The update of the weights in AdamW looks like:



Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

```
1:  given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
2:  initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment
    vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
3:  repeat
4:      $t \leftarrow t + 1$
5:      $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$      ▷ select batch and return the corresponding gradient
6:      $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1})$ $+\lambda\boldsymbol{\theta}_{t-1}$
7:      $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$      ▷ here and below all operations are element-wise
8:      $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
9:      $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t/(1 - \beta_1^t)$      ▷ $\beta_1$ is taken to the power of $t$
10:     $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t/(1 - \beta_2^t)$      ▷ $\beta_2$ is taken to the power of $t$
11:     $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$      ▷ can be fixed, decay, or also be used for warm restarts
12:     $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t/(\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) +\lambda\boldsymbol{\theta}_{t-1}\right)$
13: until stopping criterion is met
14: return optimized parameters $\boldsymbol{\theta}_t$
```

Figure 3.15: AdamW Algorithm

$$\theta_t \leftarrow \theta_{t-1} - \alpha\frac{\beta_1 m_{t-1} + (1-\beta_1)(\nabla f_t + \lambda\theta_{t-1})}{\sqrt{v_t} + \epsilon}$$

So, we’ve used AdamW in our experiments which have resulted in faster and better convergence of our model.

3.2.7 One Cycle LR

Cyclic learning rate is a learning rate scheduling technique for faster training of a network. Cyclic learning rates have an effect on the model training process. Cyclic learning rates combine the fastest possible learning rate for the discovery of the true minimum. Cyclic learning rate scheduling combines the discovery of maximum practical learning rate with learning rate annealing.

3.3 Development Steps

We started our experiments using **TensorFlow,** but the model we were using (Connectionist Temporal Connectionist (CTC)) was not fully supported in TensorFlow so we had to resort to using PyTorch. PyTorch provides CTC Loss function natively, and it also provides functions for Audio processing under module `torchaudio`.

Because of the lack of processing power and computing resources, and also because of lack of state-of-the-art training systems locally, we couldn't use the complete training dataset (about 10 GB of labeled speech samples). The full range of the training data we have was crowdsourced from Google [(http://www.openslr.org/43 and https://www.openslr.org/54)]. We only used **30,000** samples for the experiments.

9000 Samples of 5-second audio clips were used to train our model. We preprocessed them using the pytorch library. After that we fed it into our model using a free student version of Azure Machine Learning facility. Since that ran out, we had to resort to using Google Collab, which is less powerful and prone to losing all work unless meticulously and periodically checkpointed.

For the first two experiments, we tried to train the model to extract 128 features. But the CTC loss didn't stabilize below zero and instead converged to NaN. So, we tried to train for only 81 features. But finally, what did help in controlling the descent of CTC to NaN was by using Gradient Clipping of 2 instead of the default.

By looking at our base paper, we have taken 72 characters in the Devanagari script and the labels were changed to integer values.

3.3.1 Dataset analysis and preprocessing

The dataset we were using has about 157k utterances with a mean length of about 4 seconds, all recorded with a quiet background with mobile voice recorders. We discarded all samples with length more than 5 seconds, which led to us discarding about 10% of the utterances. The dataset features both male and female speakers with varying accents.

All utterances in the dataset below 5 seconds were padded to be exactly 5 seconds to make the tensor shapes consistent. We used the module torchaudio to achieve this result.3.3.2 Steps in the model

- For each utterance, its MFCC is generated and coefficients up to order 81 are selected. By consulting other research papers, and by conducting our own research, we found that coefficients beyond this point were not relevant for ASR. The input is now an MFCC spectrogram
- The labels are also preprocessed. We selected 71 different characters in the Nepali Dataset, including numbers and ignored all other characters (punctuation and devanagari characters which don't have a phonetic representation in Nepali Language). The ignored characters are: ZWNJ, ZWJ, '़', 'ॉ', '।', etc. Composite characters like 'ऋ and 'ॠ' are broken down into its constituent syllables रि.
- The spectrogram is then fed into the CNN1D layer.
- The output from CNN 1D is fed to a RNN Layer (LSTM) in our case.
- The output from RNN (LSTM) is further passed into a fully connected layer which softmax's the output and gives a probability distribution.
- The probability distribution is fed into CTC loss and the calculated loss is back propagated.

Some important notes to be made about our model:

1. Batch size was selected to be **64**
2. **Normalization layers** are used in CNN1D and Linear Layers.
3. Activations between the Layers are GeLU

4. **Gradient Clipping** was used to tackle extreme gradients
5. The Optimizer used was AdamW.
6. 2 Epochs were used to create the model but were later changed to 10 epochs.
7. **80% data was used for training** and 20% validation data was used.
8. The model has **4.9 million** parameters all trainable.
9. We've now used **OneCycleLr** (Super convergence) in our project. The Learning Rate scheduler was experimented between ReduceOnPlateau and OneCycleLR, and finally **OneCycleLR** was used.

3.3.2 Experiments for model selection and hyperparameter tuning

We performed several experiments on the dataset using various combinations of different state-of-the-art techniques and by implementing new findings from recent papers on Deep Learning and Speech Recognition. We have summarised our experiment phases and key findings below.

3.3.2.1 Exploratory experiments

Our first experiments were done with the express purpose of establishing a baseline and setting up expectations for future experiments, as well as to familiarize ourselves with all the tools we'd be later using. Our tentative goal was to take the CTC loss down to 2 if possible, and continue hyperparameter tuning. Only 9000 samples from the dataset were used at this stage. A library was used to right pad the audio below 5 seconds with zeroes and ignore the audio above five seconds. The 5 second time frame was chosen after some research and experimentation on the optimal audio length for accurate ML training.

Learning rate	Features Extracted	Epochs	Loss	Training time	Gradient Clipping	Remarks
0.01	128	2	nan	-	1	Loss goes from 3.15 to NaN
0.001	128	2	nan	-	1	Loss goes from 3.16 to NaN
0.001	81	2	nan	-	1	Loss settles to ~ 5.15, then shoots to NaN

0.001	81	2	nan	-	0	Wildly fluctuating loss, from 3 to 200, then to NaN
0.001	81	2	3.501	6:35	2	NaN issue solved due to gradient clipping
0.001	81	2	3.524	6.28	2	Redoing the same experiment with a different dataset for confirmation.

Table 3.1: Experiment 1

3.3.2.2 Trying to achieve smaller losses by adjusting input factors and hyperparameters.

After we'd verified that the runoff of loss to NaN was not a problem we tried to gain a more stable and smaller loss by adjusting other factors. First we expanded the output character set by one character and reran the experiments to prove the tentative hypothesis that an empty ϵ character was required in the CTC output to help with CTC collapsing. The experiments below proved our hypothesis true.

We also settled on the learning rate of 0.001 and 81 features extracted was important to yield the best results based on our own experiments and by reading several research papers on MFCC feature extraction and similar projects.

Epochs	Loss	Time taken	Gradient Clipping	Dropout	Remarks
2	5.476	13:23	2	0.1	Loss ranged from 3- 6 and was fluctuating
2	2.655	14:14	1	0.2	A small change in loss, need to remove punctuations from the model..

Table 3.2 Experiment 2

After removing punctuation marks and other irrelevant characters from the character set, the loss decreased further:

Epochs	Loss	Time taken	Gradient Clipping	Drop out	Remarks
2	2.3616	13:07	1	0.2	A small change in loss, need to remove punctuations from the model..

Table 3.3 Experiment 3

After this palatable loss of 2.36 was achieved we decided to venture further ideas or training the dataset. The first technique we tried was batch training of the dataset. This was the loss graph for this step with a batch size of 128.

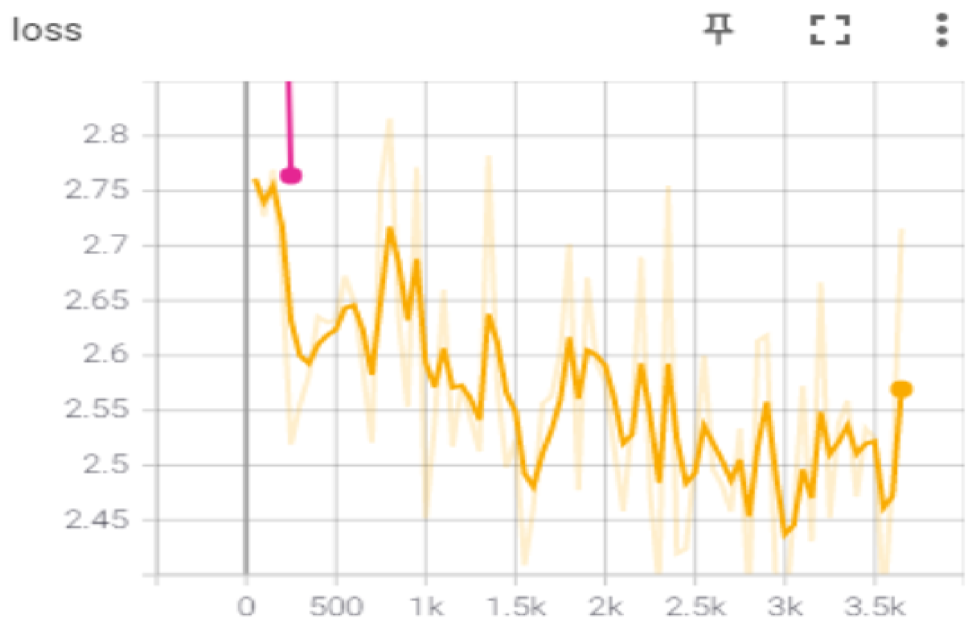


Figure 3.16: Loss graph for experiment 4.2

Epochs	Loss	Time taken	Gradient Clipping	Dropout	Batch_size
4	2.3	2:39	1	0.2	20
4	2.42	1:03	1	0.2	128

Table 3.4: Experiment 4

We then hypothesized that using SpecAugument would generalise the data better. But as it turned out, it didn't work well for our dataset with our model.

Tables occurring earlier in this report do not differentiate between validation loss and training loss because of the high and inconsistent losses observed during earlier experiments. Furthermore, since the generalizations of features are better described by validation loss, later experiments in the project were tuned by focusing on validation loss.

Epochs	Loss	Time taken	Gradient Clipping	Dropout	Batch Size	Remarks
10	1.48(training), 1.92 validation	30.22 minutes	2	0.1	20	SpecAugment was not used (version_5)
	NaN		2	0.1	20	SpecAugment was reintroduced

Table 3.5 Experiment 5

Thus we decided to not use SpecAugument for the time being.

3.3.2.3 Batch Size Selection

Batch size of 128 created worse loss than batch size of 20. So we went back to a batch size of 20. Further, after poring through the literature, we came to find that a batch size of the multiples of 8 performed better than other arbitrary sizes.

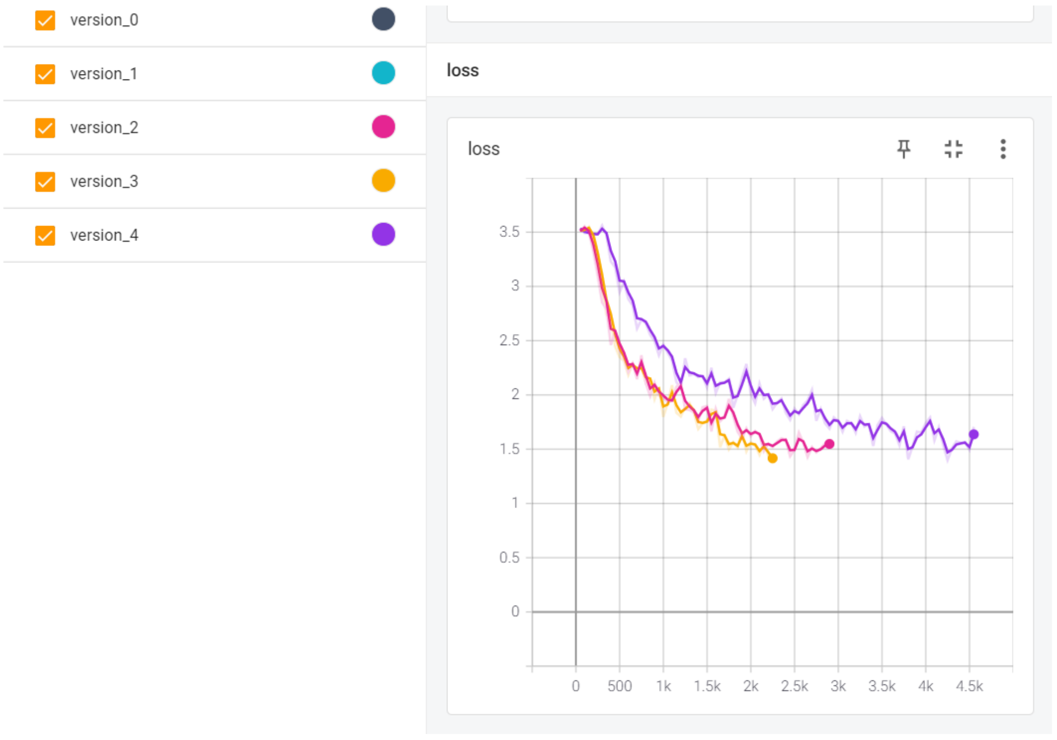


Figure 3.17: Loss visualization for batch size selection

Epochs	Loss	Batch Size	Label	CER
10	1.48 (training), 1.92 validation	20		-
	1.4 (training), 1.794 (validation)	50	version_2	-
	1.383(training), 1.6(validation)	64	version_3	0.3595

Table 3.6: Experiment 6

As is evident from the table above, batch size of 64 is the best. Other tests using batch size of 32 and 23 yielded loss so bad, we discarded the results entirely.

Finally, after all other parameters were tuned, we decided to switch our scheduler from ReduceOnPlateau to OneCyclerLRScheduler which keeps the learning rate as a variable and on training, changes the learning rate parameter from 0.01 to 0.001 to train the system.

Epochs	**Loss**	**Batch Size**	**Label**	**CER**
17	1.48 (validation),	64		0.295

Table 3.7: Experiment 7

Validation loss is saved when the loss observed in the latest epoch is lower than the currently saved loss. In the table above, the experiment was 20 epochs long but the lowest loss was observed in the 17th epoch. The records reflect this.

3.4 Tools and Techniques Used

1. **Tensorflow**

   We first used Tensorflow to train our model, but after a few experiments it was evident that the model that we had decided to use wasn't well supported on Tensorflow. Thus, we decided to not use it for the final project. However, it did provide us with valuable experience for later steps.

2. **Pytorch** (and Pytorch Lightning)

   Because Tensorflow lacks some very important tools for us (CTC loss and inbuilt audio processing features) we decided to use Pytorch. Pytorch is a lower level deep learning library, and it had all the primitives and functions we required. Pytorch lightning is a pytorch wrapper for high performance and AI research. It makes it easy to conduct experiments.

3. **OpenSLR**

   OpenSLR is used to host open-source datasets from all round the globe. We found the crowdsourced voice training samples from this website.

4. **React.js, Node.js** (for the crowdsourcing webapp)

   To build the frontend and backend of our website we used React.js and Node.js.

5. **Google Colab**

   We have used Google Colab in collaboration with Google Drive to train and tune our model.

6. **Azure Machine Learning Facilities**

   We were using Azure for training with our dataset, but because of its buggy support for PyTorch we decided to not use it for our final training.

7. **Kaggle**

   Kaggle provides 36 hours of free compute time after we upload the dataset.

8. **Tensorboard**

   To visualize the experiments directly from the logs. Interactive graphs etc etc.

9. **CUDA**

   To use the GPUs for training and parallelize the computation.

4. CONCLUSIONS

4.1 Results

The screenshot below shows our Dhwoni app (hosted at dhwoni.com at the time of this report's publication) in action. One of the authors spoke into the app's microphone and it predicted the text as \`समय\`.

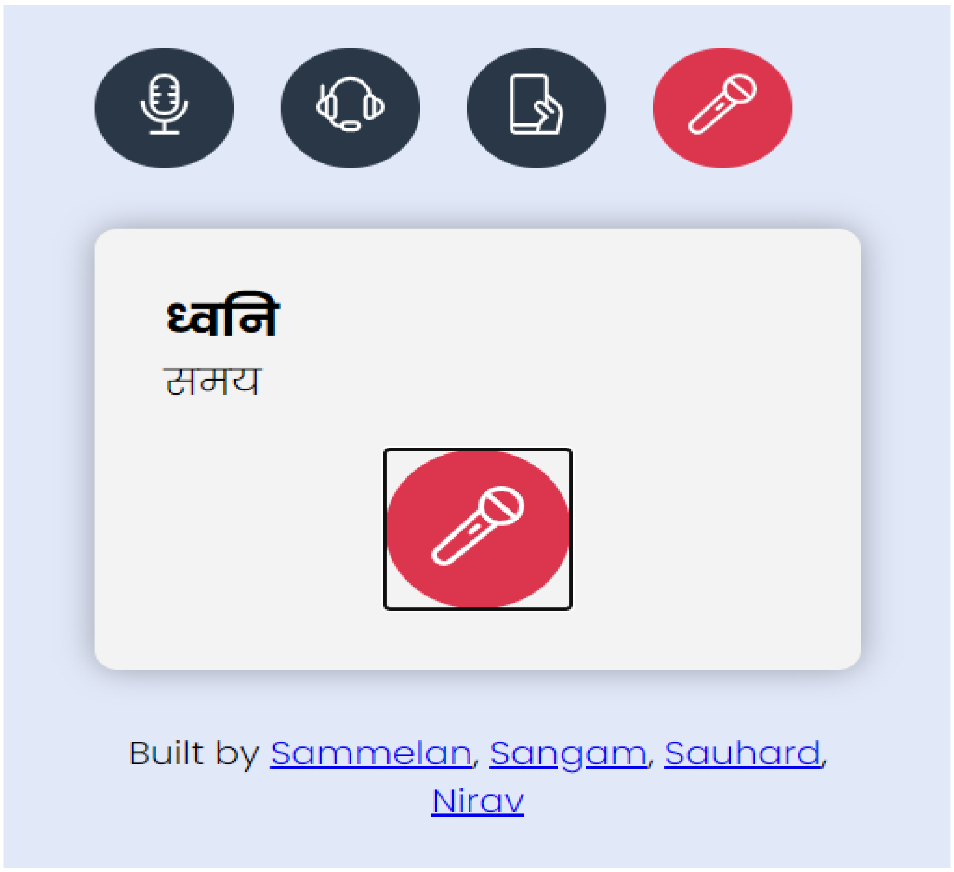


Figure 4.1: Dhwoni website

Some other interesting predictions are listed below. We spoke into the app and it transcoded it as listed.

Spoken text	Predicted Text
समय	समय
मेरो नाम संगम हो	मेरो नाम सम्गम हो

नेपालमा धेरै जिल्लाहरु छन्	मेपालमा धरे जिल्लाहरु छन्
नमस्ते नेपाल	ममस्ते नेपाल
क ख ग	क ख ग
खोला बग्दै जान्छ	आला वगै जान्छ
जिवन दुखि छ	जीवान दुखि छ
यो संसार दुखि छ	तन्चारकोस्कोमला पसन्चार कस्तोअला
व्रिन्दवासिनी मन्दिरको हजुरआमा	पराषीने मन्दिरपअधिलाम
हे अगस्थ्य मुनि, तहाँ उपरान्त ईन्द्रजिले भन्नुभयो	िव१स्टेमिन्तहाउकरण्त इनरीलेमन्ुभयो

Table 4.1: Results

In the table above, the authors spoke into the application and our system transcribed it. It should be noted that none of the authors' speech was used to train the model. Even so, the model makes accurate predictions. So it is reasonable to conclude that the model is sufficiently generalized for general use. It should also be noted that the model succeeds with smaller phrases while it struggles with longer phrases.

4.2 Discussion

Our model was able to achieve a CER of 29.5%. While it is a respectable outcome, it is nowhere near usable in the real world. In particular, it's noise resilience is terrible. We think the main reason for such poor performance is that the dataset we were using was entirely recorded inside a quiet room. So the neural network was not able to learn to accurately decouple noise from signals.

It should also be noted that less than 20% of the dataset was used due to computing and memory restrictions on our platform. By expanding the model and training with a larger portion of the dataset, better CER can be achieved.

Further, using a language model can help produce usable results from inaccurate raw transcriptions.

4.3 Further Improvements

We tried our best with the current resources to create a usable ASR system. But several areas leave a lot to be desired. We could use a bigger dataset and create a more elaborate model to accommodate the dataset. Also, further experimentations could have been done using stacked LSTMs, GRUs etc. We could preprocess the labels better to replace numbers with phonetic transcriptions. Or, we could incorporate a language model. Finally, an entirely new architecture: Transformers could be used which have shown a great promise in 2020. [7]

5. REFERENCES

[1] K. Sodimana *et al.*, “A Step-by-Step Process for Building TTS Voices Using Open Source Data and Frameworks for Bangla, Javanese, Khmer, Nepali, Sinhala, and Sundanese,” no. August, pp. 66–70, 2018, doi: 10.21437/sltu.2018-14.

[2] L. Muda, M. Begam, and I. Elamvazuthi, “Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques,” 2010.

[3] O. Abdel-Hamid, A. R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, “Convolutional neural networks for speech recognition,” *IEEE Trans. Audio, Speech Lang. Process.*, vol. 22, no. 10, pp. 1533–1545, 2014, doi: 10.1109/TASLP.2014.2339736.

[4] A. Madsen, “Visualizing memorization in RNNs,” *Distill*, vol. 4, no. 3, p. e16, Mar. 2019, doi: 10.23915/distill.00016.

[5] A. Hannun, “Sequence Modeling with CTC,” *Distill*, vol. 2, no. 11, p. e8, Dec. 2017, doi: 10.23915/distill.00008.

[6] D. Amodei *et al.*, “Deep Speech 2: End-to-End Speech Recognition in English and Mandarin Baidu Research-Silicon Valley AI Lab *,” pp. 1–28.

[7] Y. Wang *et al.*, “TRANSFORMER-BASED ACOUSTIC MODELING FOR HYBRID SPEECH RECOGNITION Nara Institute of Science and Technology , Japan Acoustic Modeling Using Transformer Experiments,” *ICASSP 2020 - 2020 IEEE Int. Conf. Acoust. Speech Signal Process.*, no. 1, pp. 6874–6878, 2020.

[8] D. Hendrycks and K. Gimpel, “Gaussian Error Linear Units (GELUs),” pp. 1–9, 2016.

[9] I. Loshchilov and F. Hutter, “DECOUPLED WEIGHT DECAY REGULARIZATION.”