Sign Language Recognition using Deep Learning methodologies

A Dissertation Submitted to the University of Hyderabad
in Partial Fulfillment of the Degree of

Master of Technology
in
Artificial Intelligence

by

Deepak Singh
19MCMI24

School of Computer and Information Sciences
University of Hyderabad
Gachibowli, Hyderabad - 500 046
Telangana, India

November 29, 2023

CERTIFICATE

This is to certify that the dissertation titled, "**Sign Language Recognition using Deep Learning methodologies**" submitted by **Deepak Singh**, bearing Regd. No. **19MCMI24**, in partial fulfillment of the requirements for the award of Master of Technology in Artificial Intelligence is a bonafide work carried out by him under my supervision and guidance.

The dissertation has not been submitted previously in part or in full to this or any other University or Institution for the award of any degree or diploma.

Dr.P.S.V.S Sai Prasad
Project Supervisor
School of Computer and
Information Sciences
University of Hyderabad.

Dr. Chakravarthy Bhagvati
Dean
School of Computer and
Information Sciences
University of Hyderabad.

DECLARATION

I, **Deepak Singh**, hereby declare that this dissertation titled, "**Sign Language Recognition using Deep Learning methodologies**", submitted by me under the guidance and supervision of Dr.P.S.V.S Sai Prasad is a bonafide work which is also free from plagiarism. I also declare that it has not been submitted previously in part or in full to this University or other University or Institution for the award of any degree or diploma. I hereby agree that my dissertation can be deposited in Shodganga/INFLIBNET.

A report on plagiarism statistics from the University Librarian is enclosed.

Date:

Name: Deepak Singh

Regd. No. 19MCMI24

//Countersigned//

Signature of the Supervisor(s):

(Dr.P.S.V.S Sai Prasad)

Acknowledgements

It gives me an immense pleasure to thank all the people who have helped me in making this project a real learning experience.

First and foremost, I would like to thank and express my heartfelt gratitude towards **Dr.P.S.V.S.Sai Prasad sir**, for his valuable guidance. Without his excellent guidance and motivation during these tough time, this project would not have been possible. It was a real privilege and great learning to work with him.

Also , I would like to thank my colleagues **Aditi jain** and **Safala chinnapaka** for their great support and motivation through out this project,without them it would have been a tough journey,especially at this pandemic time.

Last but not the least, I would like to express my heartful wishes to my beloved parents whose gracious solicitude helped me to complete this project successfully.

Deepak Singh

Abstract

According to the Indian Census 2011, one percent of the Indian community has Hearing Impairment,i.e. 1.36 crore people in India can not perform the basic need of daily life, COMMUNICATION. This makes them disconnected from the other community since a majority of people are unable to understand their language of deaf i.e., the Sign Language.

A lot of research has already been done to devise a system that can recognize and translate the sign language, and Deep Learning Methodologies have been found effective for the same cause.

This work aims at designing and developing novel Deep Neural Networks models for Sign Language Recognition which includes both **Static sign language** recognition (through images as a dataset) and **Dynamic sign language** recognition (through Videos in real-time).

We've Proposed a series of Deep Learning Architectures that are on par with earlier proposed work and have the ability to advance further if adequate data and sufficient computing power is available. Following are the Models we've proposed:-

- **Static sign language**:-
  - Optimized Deep Neural network.
  - Optimized Convolutional neural network.
- **Dynamic Sign language**:- We've developed the Dynamic Sign language recognition part into 2 separate categories:-

- Sign to GLOSS:- the signs represented by a sequence of frames per video will be converted into a collection of important words which will be used to describe the sentence in the sign language.
- GLOSS to Text:- where these identified glosses will be used in the expanded text sentence.

For both parts we've proposed and implemented the following models;-

- Sign to GLOSS:-
  - LSTM model
  - Attention model
  - Transformer model
- GLOSS to text:-
  - Transformer model.

Using these models we intend to achieve better results while recognizing and translating the Sign language. Although due to limited computation power we had to use a small data set for our project, which still is getting comparable results, our future work would be to test the designed models with high computational power and large data sets that are available.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

One of the most important aspects of human life is communication. Without communication, we will not be able to perform most of the basic chores. But according to the Indian Census 2011, One percent of the Indian community has Hearing Impairment; that is, 1.36 crore people in India can not communicate like the rest of the people. This makes them disconnected from the other community.

To make them unable to communicate, they have a special form of communication, the **"Sign Language"**.

1.1 Sign Language

Sign languages are a combination of hand gestures, where for each alphabet, words there are special predefined hand movements.

These hand movements, sometimes accompanied by facial gestures, hand positions, and hand shapes, are used by sign language users in order to express themselves to others (either to peoples with special abilities (non-speaking, deaf community) or regular people).

These hand gestures can include one or both hands at a time, depending on the word to be conveyed.

Based on these hand movements **Sign languages** can be organized into 2 categories[33]:-

- Static sign language
- Dynamic sign language

1.1.1 Static sign language

Sign languages where there's no continuity in hand movement,i.e., the hands are not in motion.These type of sign languages are mostly used to represent simpler signs.Some example of these type of signs are as follows:-

- **Representing numerals and alphabets in sign language**:- These are the most simpler static sign languages, where to represent a numerical digit or an alphabet in a sign language we use static sign languages.

Figure 1.1: Static sign for Numeral digit 2

Figure 1.2: static Sign for alphabet 'A'

1.1.2 Dynamic sign language

Dynamic sign Languages include movements of hands in order to generate a valid signature for a particular word or phrase in a sentence. These sometimes are incorporated with facial expressions as well.

1.2 Motivation

Even though the special community of peoples with hearing or speaking impairments have a form of communication, that is the **Sign language**, they still can not communicate with the other community without these special abilities.

The reason being most of the peoples are not able to understand these signs, they are unable to comprehend what the other person (using sign language) want's

to convey. That is because they don't have the knowledge of sign language. After all, unlike verbal communication, these include various hand movements and facial expressions, which sometimes are confusing to understand due to similarities between them.

Through this project, we would like to fill this major communication gap by building a system that would be capable of understanding these signs and can convert them into a textual format understandable by other persons.

1.3 Deep Learning

Deep Learning is a paradigm of Machine learning, where it learns the task at hand with the help of a huge dataset as a training tool. Deep Learning models accomplish this task by mimicking the working of the human brain, where a human brain consists of numerous amount of neurons that pass the information throughout its network in the form of an electrical signal. A deep learning model also does the same,i.e., passes some kind of information throughout the network and makes the model learn extraordinarily, and tends to make very good decisions while learning a task.

Deep Learning models are proved to be very efficient while learning very complex tasks such as **Natural languages processing and image processing**, the reason for this is the availability of a huge amount of **data** which can be used for the training purpose of the model. Apart from that, a huge number of **Parameters and Hyperparameters** makes it feasible for Deep learning models to learn a complex task.

1.4 Problem statement

The objective of our project is to devise a novel Deep Learning Models that can improve the accuracy in recognizing the meaningful signs from Static and Dynamic sign languages and convert them into understandable sentences.

1.5 Organization of dissertation

The entire dissertation is organized as follows:

- Chapter 2. Describes some of the **Deep learning** architectures that have been proved vital for our project, along with some of the terminologies related to these architectures.
- Chapter3. Literature that we've reviewed in order to gain a better understanding of **Static sign languages**. This is followed by some of the architectural designs we've proposed.
- Chapter4. Literature review for **Dynamic Sign languages**, followed by some architectures we are proposing.
- Chapter 5. Implementation details of the architectures proposed for both **Static sign languages** and **Dynamic sign languages**.
- Chapter 6. This includes results we've obtained for our experiments and details of Data sets we've used for our project
- Chapter 7. The future work and the conclusion are given in this chapter.

Chapter 2

Deep Learning Architectures.

We will discuss some of the fundamental Deep learning models and terms related to them that have been proved vital for our project.

2.1 Deep Neural Networks

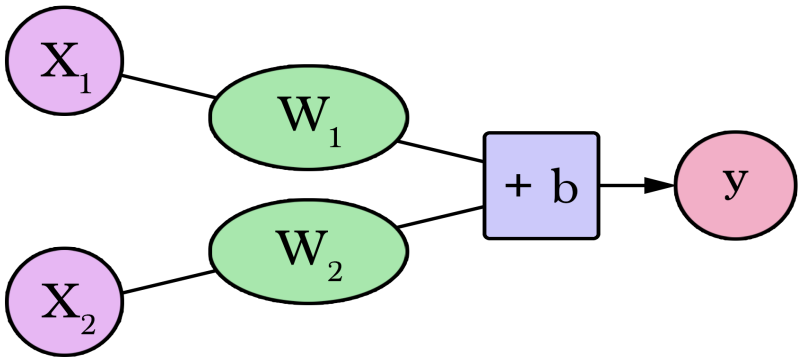

Figure 2.1: simple deep neural network

Deep Neural Networks[17] are the most basic architecture in Deep Learning. This model consists of a number of layers having a lot of **Nodes** where each **node** represents a particular feature of a data set, for example, while predicting a housing price, the number of rooms in a house can be of importance and hence is a feature. These features are incorporated with some parameters like **weights, bias**, and hyperparameters like **activation functions, learning rate, optimizations, etc.** which provides much-needed functionality in order to help

the model to predict the correct outcome based on these features, parameters, and hyperparameters. Deep Neural Networks has the following major components-

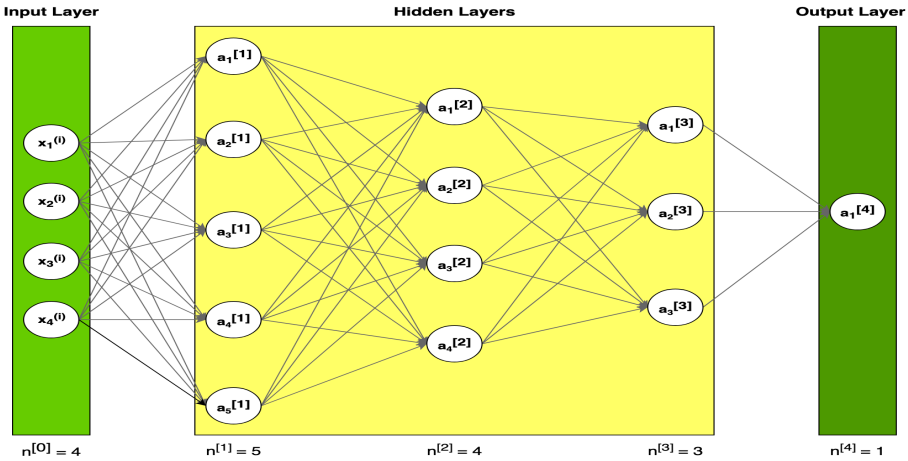

Figure 2.2: Deep Neural network

- **Input Layer**:- An input layer consists of **N** number of nodes, representing **N** features of a data set.

- **Hidden layers**:- The model consists of many hidden layers (usually in the range of 1 to 10), but in some architectures, it can go as deep as 150 layers.

  These layers consist of nodes similar to the input layer but represent features after passing the input through some nonlinearity functions, for example, **Sigmoid** or **tanh** in order to get better results.

- **Activation functions (A)**:- Activation functions[10] are some mathematical computations that help to make the features more susceptible and robust to fit the desired output. Some of the Activation functions are as follows.

  - **Sigmoid activation function**
  - **tanh activation function**
  - **Relu activation function**
  - **Softmax activation function**

- **Weights (W)**:- Weights are the parameters that are a measure of the influence a particular feature can have while learning a task and giving some outputs. These **Weights** are learnable in nature and can be recomputed

during **forward** as well as **backward propagation** based on the accuracy of output we are getting.

- **Bias (b)**:- bias is learnable parameters that have a constant value associated with them. These biases help to shift the activations by a constant amount to fit the learning curvature.

- **Forward pass**:- During the training stage, the features after applying some activation functions are sent to the next layer in the model in order to get better feature representation; this process is known as a forward pass. During this pass, an intermediate value (Z) is calculated using weights(W) and bias(b) and previous layer activation values, then this intermediate value is passed through the activation function ($g^{[l]}$) to calculate activation values(A) of next layers.

$$Z^{[l]} = W^{[l]} \times A^{[l-1]} + b^{[l]} \tag{2.1}$$

$$A^{[l]} = g^{[l]}(Z^{[l]}) \tag{2.2}$$

- **Loss Functions (J)** :- Loss functions helps to calculate the error in predictions w.r.t actual values. High loss implies less accuracy, whereas low loss implies high accuracy.

$$J(W, b) = -\frac{\sum(y^{(i)} \times \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \times \log((1 - \hat{y}^{(i)})))}{m} \tag{2.3}$$

Where, $y^{(i)}$ are the actual expected output, for a layer i and $\hat{y}^{(i)}$ is the predicted output for a layer I, and m is the total number of nodes on a particular layer.

- **Backward pass or Back-propagation**:- After forward pass, if the error calculated by loss functions is high, then the model needs to re-calibrate its parameters (weights, biases) by sending back the gradients of loss, thus updating its parameters during backward pass.

$$W = W - \alpha\frac{\partial J(W, b)}{\partial W}, b = b - \alpha\frac{\partial J(W, b)}{\partial b} \tag{2.4}$$

During this back-propagation, we will use some of the optimizers functions like **Gradient descent[24], Stochastic Gradient Descent[23], and Adam optimizer[16]** which will help us to compute the loss function gradients with respect to the parameters **Weight and bias** and in case of these gradients being too high, the backward pass will help to decrease these gradients based on a hyperparameter **learning rate**, thus managing the gradients and eventually optimizing the learning process.

- **Output layer (Y)**:- The output layer contains nodes equal to a number of the desired output, where each node represents a particular output.

2.2 Convolutional Neural Network

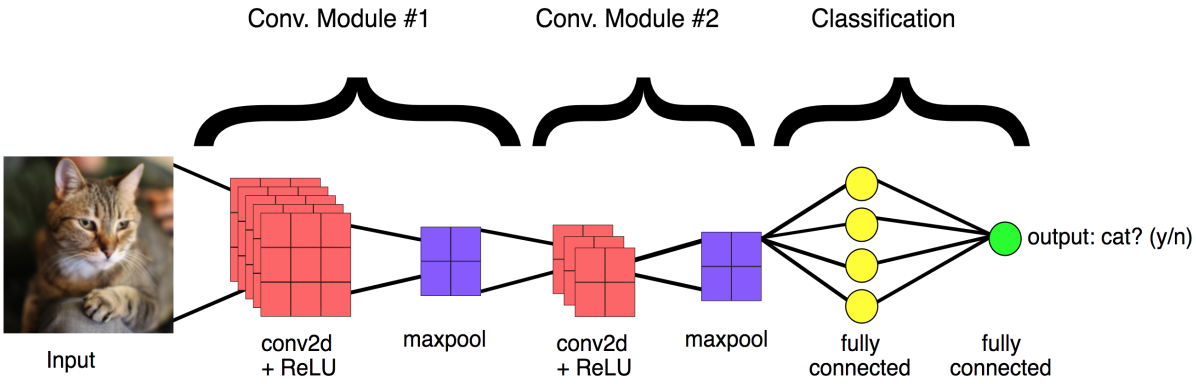

Figure 2.3: Convolutional Neural network

Convolutional neural networks[20] are the best choice when our dataset is in the form of image or video, the reason being similar feature types, i.e., pixel values ranging between 0 and 255. Convolutional neural networks can compute these parameters in parallel, starting from low-level features (maybe a horizontal line) to high-level features (a face or something more complex) for these kinds of datasets, hence reducing the number of learning parameters to a greater extent.

The main advantage of convolutional neural networks are:-

- **Parameter sharing**:- At a particular level, it will detect features with similar quality and thus will share a lot of parameters.

- **Sparsity of connections**:- In each layer, each output value depends only on a few inputs, so the rest of the inputs will not have any effect on these outputs, so this reduces overfitting cases.

A convolutional neural network uses a convolution Layer known as a **Filter**. A convolutional neural net involves a convolution operation between the inputs and the Filter. These convolved values are passed through some activation functions, and to improve accuracy, some **regularization methods** like the **dropout method or L2 Norm** are also meant to be applied.

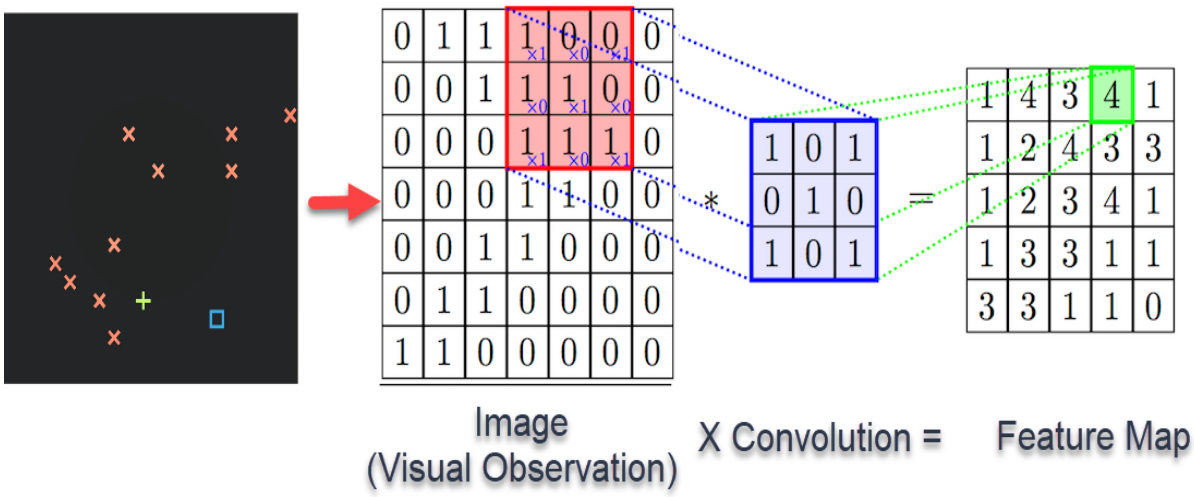

Figure 2.4: Convolution operation

2.2.1 Pooling layers

Other than convolution layers, a Convolutional neural network often consists of pooling layers. This is done for the following reasons:-

- to reduce the size of the representation of input data.
- to speed up the computations.
- makes some of the features it detects a bit more robust.

2.2.1.1 Maxpooling Layer

A max pool layer of size $\mathbf{n} \times \mathbf{n}$ will look into a picture portion of similar size and will take the maximum value in that region and discard the rest of the values,

thus reducing the size by n times.

2.3 Sequence Models

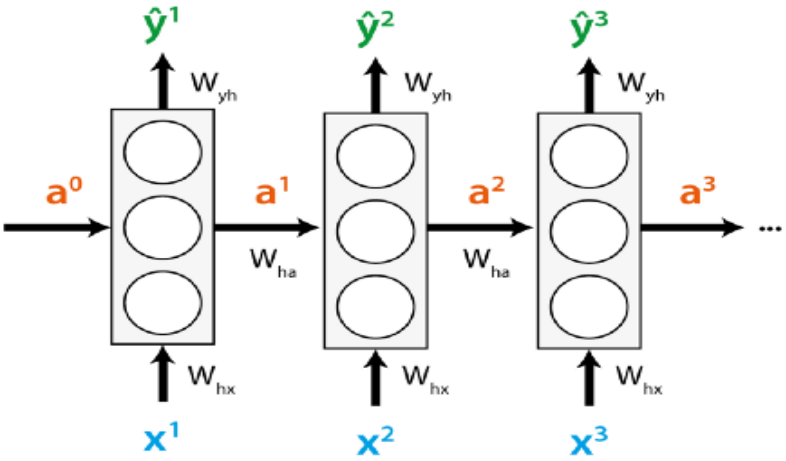

Figure 2.5: Sequence model

Sometimes our dataset consists of not only spatial dimension but also temporal dimension,i.e., it grows with time also, for example, a sentence where the sentence grows with time, and the upcoming words depend on previous words in the sentence. In these types of problems, previously discussed models are not proved to be working well. In these time-driven data, the input values vary with time, so a different model which can take care of temporal dimension is required.

The sequence model[25] can work with temporal data tremendously well.

The **Input** to the sequence models are the feature vectors of our time dimensional incorporated datasets where a **feature vector** represents some numerical format of input data per timestamp.

The **Output** by the sequence models are the **predicted feature vector** of the next timestamp, based on the following things:-

- Current timestamp input feature vector.
- Activation values of previous as well as current recurrent units.
- Weight matrix, which will be the same for all timestamps per layer of sequence models.

- **Modified inputs**:- In some sequence models like **Attention models, Transformers** etc., during the **Training phase** it might happen that the input at a timestamp is the **actual previous time stamp feature** i.e., $x^{<t>} = y^{<t-1>}$. And during the **Testing phase**, the input to the current timestamp is the **predicted features at previous time stamps** i.e., $x^{<t>} = \hat{y}^{<t-1>}$.

2.3.1 Gated Recurrent units

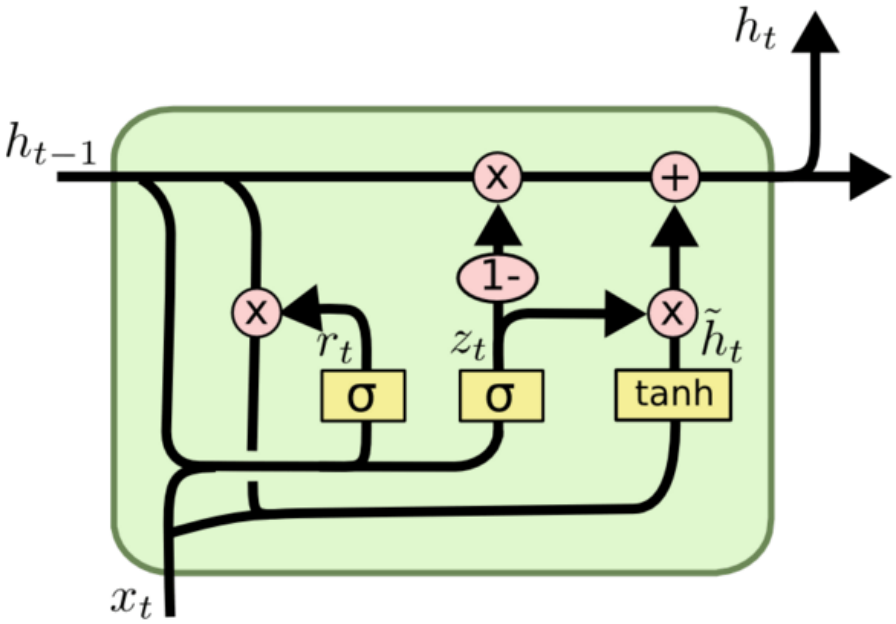

Figure 2.6: Gated Recurrent units

Gated Recurrent Neural networks[8, 7] are special kinds of networks that have a special cell used to memorize the features in the past and later use them to predict next upcoming outputs. These special cells are called **memory cells**. A GRU's unit consists of the following components:-

- **Memory cell** ($h^{<t>}$):- a memory cell memorizes the feature of input at any given instance of time **t**
- **activation** ($z^{<t>}$) :- computed activation for an input at any given instance of time t.
- **update gate**(r_u):- computes whether to update the memory cell or not.

- **candidate for updating** ($\tilde{h}^{<t>}$) :- if a memory cell needs an update then this will be the candidate value for memory cell.

A GRU is driven by the following equations at any point in time:-

$$h^{<t>} = z^{<t>} \tag{2.5}$$

$$\tilde{h}^{<t>} = tanh(W_c \times [h^{<t-1>}, X^{<t>}] + b_c) \tag{2.6}$$

$$r_u = \sigma(W_u \times [h^{<t-1>}, X^{<t>}] + b_u) \tag{2.7}$$

$$h^{<t>} = r_u \times \tilde{h}^{<t>} + (1 - r_u) \times h^{<t-1>} \tag{2.8}$$

where

- h = memory cell value
- $\tilde{h}$ = candidate value for memory cell
- σ and *tanh* are activation functions

2.3.2 Long Short term Memory

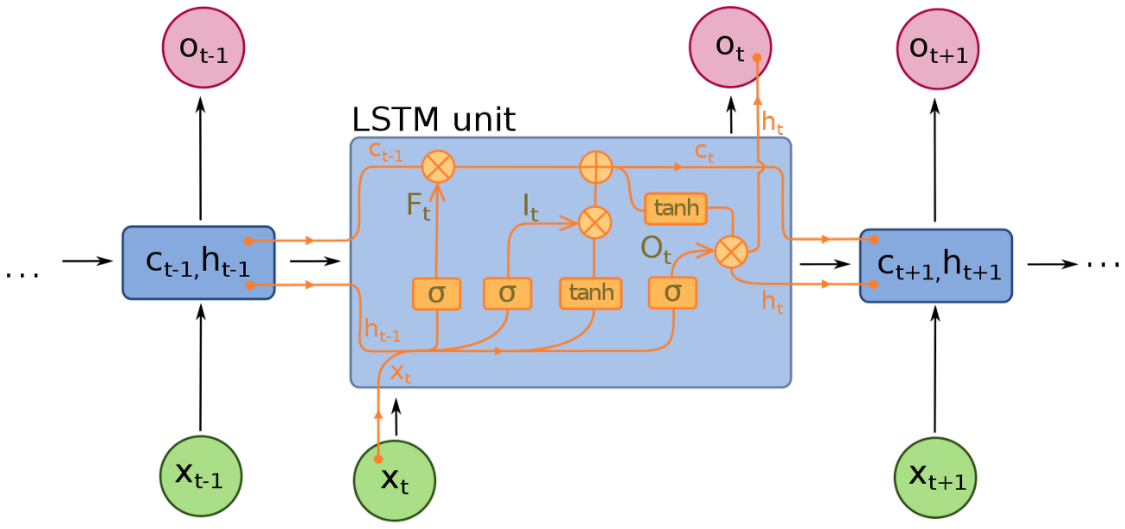

Figure 2.7: Long short term memory unit

LSTM's[12] are more general version and more Powerful version of GRU's, where it not only have a **memory cell** but also a **forget and update cell** which helps the networks emphasize in a more general way.

This forgetting property of LSTM helps it to give more generalized and better results. LSTM's are driven by the following equations:-

$$\tilde{c}^{<t>} = tanh(W_c \times [h^{<t-1>}, x^{<t>}] + b_c) \tag{2.9}$$

$$F_u = \sigma(W_u \times [h^{<t-1>}, x^t] + b_u) \tag{2.10}$$

$$F_f = \sigma(W_f \times [h^{<t-1>}, x^t] + b_f) \tag{2.11}$$

$$O_o = \sigma(W_o \times [h^{<t-1>}, x^t] + b_o) \tag{2.12}$$

$$c^{<t>} = F_u \times \tilde{c}^{<t>} + F_f \times c^{<t-1>} \tag{2.13}$$

$$h^{<t>} = O_o \times tanh(c^{<t>}) \tag{2.14}$$

where

- F_u = an update gate computes the amount, a memory cell should consider while updating its value w.r.t the candidate value. Its value ranges between **0** and **1** where a value near 0 suggests not to update the memory cell with the candidate value, whereas a value of 1 totally suggest changing the memory cell value with the candidate value.

- F_f = a forget gate computes the amount a memory cell should forget memory cell value of previous timestamps. It's value ranges between **0** and **1** where a value near to 0 suggests forgetting the previous timestamp memory cell value, whereas a value near 1 suggests not to forget previous timestamp memory cell values.

- O_o = the output gate computes the order by which the current timestamp activation value should change w.r.t current timestamp memory cell value.

- c = memory cell

- h = activation value

- x = input feature value

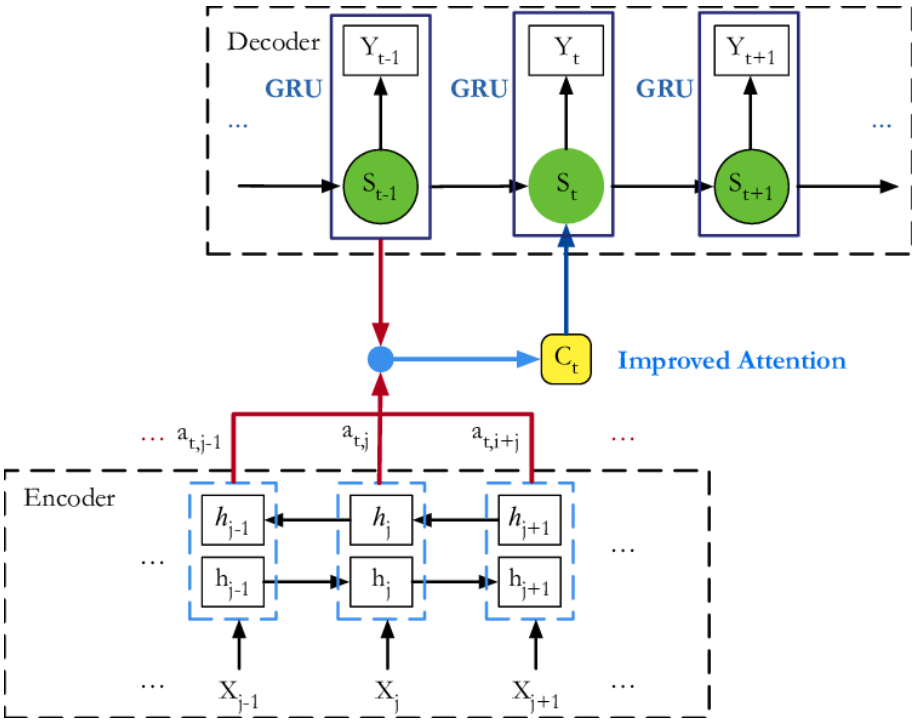

Figure 2.8: Attention model

2.3.3 Attention Models

The problem with LSTM's and GRU neural networks is that given a very long sentence, these models are unable to memorize better representations and hence are not able to generalize well.

To cope up with this problem, Attention models[3] came into the picture, where the model consists of a special **Attention layer** which helps to allocate some weights based on computed values to each feature throughout the sentences, and according to these attentions, the decoder will generate the outputs.

The Attention Model consists of 3 major components, namely:-

1. Encoder layer
2. Attention layer
3. Decoder layer

2.3.3.1 Encoder Layer

Encoder layer is similar to other Recurrent Neural Networks,where a layer consists of **n** LSTM unit or GRU unit,where **n** is the number of input data size.

Each LSTM unit will have multiple hidden units which will calculate activation values w.r.t input feature values.

2.3.3.2 Attention layer

The **Attention Layer** will compute attention based on some softmax computation for each of the **n** units in the encoder layer and concatenate all the attention values in order to compute **context vector** for all the encoder layer unit.

These context vectors will act as input for **Decoder layer**.

the computations done at the attention layer are as follows;-

$$c^{<t>} = \sum_{t'} (\alpha^{<t,t'>} \times a^{t'}) \tag{2.15}$$

$$\alpha^{<t,t'>} = \frac{\exp(\mathrm{e}^{<\mathrm{t},\mathrm{t}'>})}{\sum_{t'=1}^{T_x} \exp(\mathrm{e}<\mathrm{t},\mathrm{t}'>)} \tag{2.16}$$

$$\tag{2.17}$$

where

- $c^{<t>}$ is the context vector for any time stamp t of decoder unit
- $\alpha^{<t,t'>}$ are the attention weights for time stamp (t) of decoder layer w.r.t (t') of encoder layer
- $\mathrm{e}^{<\mathrm{t},\mathrm{t}'>}$ are **Energy variables** computed with the help of a simple deep neural network,taking activation of encoder and decoder as input.

2.3.3.3 Decoder Layer

A Decoder layer is identical to the Encoder layer. The only difference is, instead of taking input directly from the data set, it will take input from the attention layer along with previous timestamps output.

2.3.4 Transformers

Transformers[32] are the most powerful models discussed so far. It is a generalized version of the attention model, where the whole computation in both encoder

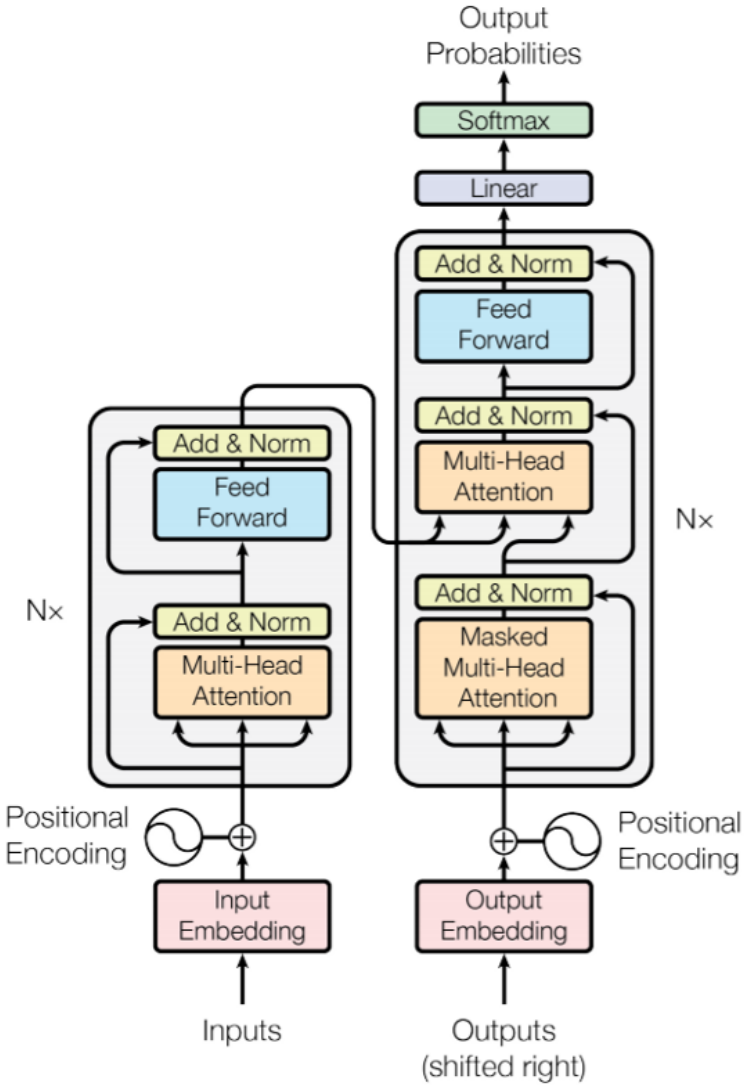

Figure 2.9: Transformer model

and decoder is done in parallel, thus reducing the computation time to a greater extent.

The transformer comprises of following components:-

1. **Input Embedding**:- Sometimes the inputs are in the form of textual data, which are not a good format for machines, so in order to make that suitable, they are converted into numerical embedding vector. This can be done using an extensive dictionary and one-hot encoding of the word.

2. **Positional Encoding**:- One problem with Previous models was, they ignore the positions of input data, so if these data come in any order, the output

will always be the same. To resolve this problem, Transformer introduces a ***Positional Encoding matrix*** which provides a unique encoding to all input vectors, Thus improving the results to a greater extent.

3. **Query ,key, Value matrix**:- learnable matrices w.r.t some **Weights** and **input embedding matrix**

4. **Encoder Layer**

   (a) **Multihead Attention**:- This is the most Important Computational part of a Transformer.

   - **Self Attention**:- It is somewhat similar to the attention used in the Attention model, where for each time steps in the input data, the model will learn an attention value.

$$A(q,k,v) = \sum_i \frac{\exp(q \times k^{<i>})}{\sum_j (\exp(q \times k^{<j>}))} \times v^{<i>} \tag{2.18}$$

   where,

   - **A(q,k,v)** = attention calculated w.r.t query,key,value matrix.
   - **i** = time step for which attention is being calculated.
   - **j** = all other time steps.

   Multi head Attention will learn **n** numbers of Self attention,based on some **Query (Q),key(k),and value(b)** matrix and then concatenate them all together.

$$\sum_i (head_i) \times W_o \tag{2.19}$$

   where,

   - $head_i = i^{th}$ attention value
   - W_o = some learnable weight matrix

   (b) **Normalization and Residual networks**:- in order to avoid over-fitting,some normalization **batch normalization or regularization** along with **residual connection** is implemeted

(c) **Feed Forward Neural network**:- A final dense layer,after which the encoder will generate some intermediate representation of inputs that are passed to **multihead attention of decoder part**.

The encoder part is repeated for **N** times.

5. **Decoder Layer**

   (a) **Masked Multihead Attention**:- Since during testing phase, getting input at time stamp **t** the decoder's needed to output the next timestamp output, so in order to train it to do so, some of the input embedding parts are masked by junk values during training, so that it will be tough for the transformer to guess the output and hence it will learn properly.

   It is also a form of **normalization**

   (b) **Multihead Attention**

   (c) **Normalization and Residual networks**

   (d) **Feed Forward Neural network**

2.4 Hyperparameters

Hyperparameters are some extra parameters that a Deep Learning/Machine learning model has, along with some mandatory parameters.

These hyperparameters are not necessarily needed to be included in a model, but if included, they can help a model learn better and give better results.

These hyperparameters can also be fine-tuned separately, sometimes using optimizations methods, sometimes manually by human interference, in order to make a model learn better parameters. The following sub-sections will discuss some of these important hyperparameters.

2.4.1 Activation function

An activation function[10] is a mathematical function that provides non-linearity to the model and helps it learn better curves while looking at the features of a

data set.

Some important activation functions are:-

- **Sigmoid**:-

$$a = \frac{1}{1 + e^{-z}} \tag{2.20}$$

- **tanh**:-

$$a = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.21}$$

- **Relu**:-

$$a = max(0, z) \tag{2.22}$$

- **Softmax** :-

$$a = \frac{e_i^z}{\sum_i e_i^z} \tag{2.23}$$

where,

- a = activation value
- $z = W \times x + b$
  - w = weight matrix
  - x = features
  - b = bias vector

2.4.2 Learning rate (α)

learning rate is a hyperparameter, which decides the speed at which a gradient should increase or decrease for modifying the parameters (**W, b**) in case of high error.

For most cases, we use a manual learning rate, for example, 0.001,0.003, etc. However, for Transformers, we have used a custom Learning rate as described in "Attention is all you need[3]."

2.4.3 Loss function

Loss function helps in tracking the distance between the predicted output and actual output; the models' purpose is to decrease this distance, i.e., the loss, in order to optimize the results.

some of the losses we have used are

- **Logarithmic loss function**:-

$$J(W,b) = -\frac{\sum(y^{(i)} \times \log(\hat{y}^{(i)}) + (1-y^{(i)}) \times \log((1-\hat{y}^{(i)})))}{m} \tag{2.24}$$

- **Linear loss function**:-

$$J(W,b) = -\frac{\sum\left(\frac{(\hat{y}-y)^2}{2}\right)}{m} \tag{2.25}$$

2.4.4 Optimizer

Optimizer is a function that helps to manage the parameters like **Weights and bias** in case of high loss. The job of optimizers is to keep the loss at a minimum.

some of the optimizers we have used are

- **Gradient descent[24]**
- **Adam optimizer[16]**.

2.4.5 Regularizers

Some times it might happen that a model is overfitting due to the following reasons:-

- Less training data but a high number of features.
- unbalanced training data.

In order to help the model train better under these circumstances, we use normalization or Regularization methods. Some of the regularizations we used in our project are:-

- **L2 norm**:- l2 norm increases the weight to a very high amount based on a Hyperparameter λ, thus minimizing the effect for a particular feature that is causing the model to overfit.

- **Drop out method**:- it randomly zeros out a proportion of weights on a particular layer so that a model should not rely on a particular feature and always spread out its weight to all the features available.

2.4.6 Sampling method

When using time sequence data, we need some sampling method that will sample the predicted output for a current time step and will select the most suitable output based on the probability of generating that output.

So, we have used the following sampling methods:-

- **Greedy Sampling** :- The Greedy sampling method is the most simpler sampling method available in the literature.

  This sampling method works purely based on the highest probability of the words; that is, out of our whole vocabulary, it will predict the word which will have the highest probability based on the input features.

  At each timestamp, it will use the greedy method to pick the words and append them to the generated sentence, which will be our final output.

The problem with **Greedy sampling approach** is that every time it selects the outcome based on the highest probability, which might not be suitable every time.

Since In our dataset we have stuffed all the text in gloss as well as German text with $< startseq >$, which is a flag for the model to denote the start of a sentence, this sampling method was predicting the same sentence for over 90% of the time, which is clearly not suitable.

To resolve this issue, we used a more complex and better sampling method **Top -k sampling method**.

- **Top -K sampling method**:-This sampling method[13] makes sure that the words with less probability than the **top k probabilities** should not be considered for generating the output.

  After selecting the top-k possibilities based on the highest probability, it will redistribute the remaining probability to only these **k** samples, and out of these, it will select **one** output at random, thus decreasing the chance of the same sentence for all the input data.

For our project, we have used $\mathbf{k = 5}$ as our sampling measure.

2.4.7 Evaluation metrics

Evaluation metrics are functions that help in checking and evaluating the results obtained w.r.t results obtained by other researchers.

1. **BLEU**:-

   BLEU score[21] is used to compare the obtained text results with the available reference text.

   The BLEU score is a ratio of counts the number of times a word or a collection of words (a collection of n words is known as **n-gram**) appear in the reference text, w.r.t the word or collection of words that appear in the result text.

   The range of BLEU score is in between (**0 and 1**).where 0 represents no match at all, and 1 represents a perfect match.

   Formally,

$$BLEU_n = \frac{\sum_{n-grams \in \hat{y}} Count_{clip}(n-gram)}{\sum_{n-grams \in \hat{y}} Count(n-grams)} \tag{2.26}$$

   where,

   - n-gram:- number of words taken as a pair.

- $Count_{clip}$ = max number of times an n-gram appears in all of the reference text per data in a dataset.
- Count = a total number of times an n-gram appears in result text.

2. **WER**:-

   the word error rate is defined as a total number of mismatches of words in the result with respect to a total number of words in reference text.

$$WER = \frac{substitution + insertion + deletion}{number\ of\ words\ in\ the\ reference\ text} \tag{2.27}$$

   where,

   - Substitution = words which are replaced by other words in the result.
   - insertions = words that are additionally added in results and are not in the reference text.
   - deletions = words which are there in reference text, but not in result text.

3. **PRECISION**:-

   Precision is the numerical measure of correctness of a Machine learning model where it can be described as the ratio of the actually correct predictions by the model with respect to all the predictions which are deemed correct by the model,that is it might happen that some of the predictions which our model gave as a correct predictions are incorrect in reality.

$$precision = \frac{true\ positive}{true\ positive + false\ positive} \tag{2.28}$$

4. **RECALL**:-

   Similar to Precision, Recall is also a numeral correctness score of a model. but unlike precision,Recall is a ratio of Actually correct predictions with respect to Actually correct predictions along with All predictions which are

correct in reality but are predicted as incorrect by the model.

$$recall = \frac{true\ positive}{true\ positive + false\ negative} \tag{2.29}$$

where,

- **True positive** :- predictions which are identified as positive by model and in reality also they are positive.
- **False positive** :- predictions which are identified as positive by model but in reality they are negative.
- **True Negative** :- predictions which are identified as negative by model and in reality also they are negative.
- **False Negative** :- Predictions which are identified as negative by model but in reality they are positive.
- In the **Table** 2.1 depicted below **1** represents Positive examples and **0** represents Negative examples.

Actual Class			
Predicted Class		*1*	*0*
	1	True positive	False Negative
	0	False Negative	True Negative

Table 2.1: Model Prediction Types

5. **F1 Score**:-

F1 score was introduced to help as a deciding factor,which can measure performance of a model by keeping a balance between precision and recall measure.

$$F1 = 2 \times \frac{precision \times recall}{precision + recall} \tag{2.30}$$

Chapter 3

Static Sign Language

We have seen a lot of Deep learning architectures in **chapter 2**, which are great at learning tasks as complex as the one in hand, i.e., Sign language recognition, where our dataset consists of not only images and videos but also textual data too. In this chapter, we will discuss our tasks in terms of their complexity and some of the proposed solutions w.r.t these architectures.

Static sign language[33] does not involve any kind of movement that includes all of the body parts. But there are various other factors which is to be taken care of while understanding a static sign language. Factors involved in Static Sign language can be classified into various categories based on the following criteria:-

- Number of hand included:- whether one or both hands are included while gesturing a sign.
- Manual sign:- Signs that include only hand, i.e., they are not accompanied by any facial or body gestures.
  - hand shape
  - hand orientation
- Non Manual sign:- Signs that include both hands as well as facial gestures.
  - Mouth Gesture
  - Body posture
  - Face Expression

Based on this, there can be a lot of things we should consider while understanding a sign and using it in our project, which can be overwhelming. So in order to keep things simple, we are using Static sign language for the following only:-

- American Sign Language for Digits and English alphabets:- An Image for signs will be sufficient to understand static signs.
- Indian sign language for English alphabets.

3.1 Literature review

3.1.1 Indian Sign Language using Neural networks

In this literature[27], Basic Deep neural networks can be proved sufficient in order to identify the underlying alphabet in an image. From the images, the location of hands is identified and converted into a grayscale format by using feature extraction methods like **Hidden Markov Models**.

Then from these segmented hand shapes, each finger position is located by using Euclidean distance from the center of the palm. Measuring the angles of each finger is done based on the position of fingers from the center of the palm. This was important in order to identify the alphabets in the image.

This information is used as an input for a Deep dense neural network, which uses **logarithmic cross-entropy loss** as the calculating errors and gradient descent for optimizing metrics. The model was proved Sufficient to capture underlying features and identify the alphabets, with 89.47% precision and 89.78% recall score.

3.1.2 Static sign language recognition using Deep Learning

Although Deep Dense Neural Networks are good at capturing features in images; Convolutional Neural Networks are the best for the job due to the small number of learnable parameters and parallelism. In comparison to Deep Dense Neural Networks, this makes it very efficient in terms of computing and memory. This work[31] incorporates American sign language for English alphabets, as well as some unique signs for words like (**love, money**), etc. Only three layers of

convolution filters plus a **Maxpooling layer** were found to be sufficient for correctly identifying the alphabets. It outperformed the Deep Dense Neural Network in terms of accuracy, reaching a stunning 97.52 percent. It also reduced the time it took to recognize an alphabet from a given image.

3.2 Implemented approach

3.2.1 Deep Dense Neural network

Because no architectural changes are possible, we must rely on better optimization techniques to improve these networks. The following tactics are being used::-

- Logarithmic cross-entropy loss is chosen instead of mean squared error because it is more dependable and robust.
- The adam optimization technique is used to improve the weight matrix and, as a result, the final outcomes.
- **Regularization** and **Dropout** method is also used in order to prevent overfitting.
- **Relu activation function**:-This is one of the best activation functions due to the fact that it computes gradient descent faster even when the derivatives are maximum or minimum.

3.2.2 Convolutional Neural Network

In recent years, there has been a large amount of study on convolutional neural networks. A slew of new models have emerged, all of which are doing admirably in the field of computer vision. These are some of the models. :-

- **ResNet**[11]
- **Inception Net**[30]
- **VGG16**[28]

Even if a plethora of Advanced convolutional neural networks is available, the problem is that with an increase in complexity of the model, the number of parameters as well as hyperparameters increases exponentially. For example, the number of parameters in **VGG16[28]** is ***138 million***, which can be proved overwhelming, especially for us because we lack the computing power at this point in time, so we have decided to proceed with a Shallow convolutional network with plenty of optimizations done in order to achieve our target, So we are proposing the following Convolutional neural network model:-

- Optimizing a simple ConvNet using various optimization techniques available.

Chapter 4

Dynamic sign Language

As discussed earlier in **section 1.1.2**, Dynamic sign Languages do involve movements of hands while making a valid sign gesture. But along with that, there's a ton of other things we need to keep in mind in order to correctly identify a sign. Some of the factors involved in Dynamic sign languages are as follows:-

- Number of hands involved in making a sign.
- **Manual sign**:- that does not include other body parts like **facial expression** to make a sign. This includes;-
  - Handshape
  - hand orientation
  - hand location
  - hand movements
- **Non-Manual Sign**:- these include hands along with other body gestures to complete a sign. This includes:-
  - Mouth gesture
  - Body posture
  - Face Expression

All of these things can be proved overwhelming to a non-sign language user, but for a machine, it is possible to learn and synthesize this data in a matter of time.

For Dynamic Sign languages, the appropriate data set would be a collection of videos, where each video can represent a whole sentence. Working on these kinds of data set is a tough job due to the following reasons:-

- All videos can have different lengths, and it's hard for Deep learning architectures to work with uneven data sets.
- A sentence per video can have multiple signs in it, and it's very hard to identify the time bounds for each sign, i.e., a time frame where a sign ends and another starts.

To resolve with this problem we will have to consider **Spatial Feature** along with **Temporal Feature** of a video.

4.1 Literature review

4.1.1 Better Sign language translation with STMC-Transformer

This research by **Kayo yin**[34] was of utmost importance and an inspiration to us. This work divides the whole project into the following parts:-

- Identifying spatial and temporal features from a video:- This part of the project converts a video into a sequence of frames with 4 different crops per frame of each **Face, hands, full, pose** and uses a unique **STMC** module in order to identify the underlying **spatial features** from the video and then these features are passed along to a **TMC module** to capture the temporal features.
- Learning **Glosses** from a video instead of whole sentence from a video using **Bi-directional LSTM**:- The learned features from the STMC-TMC module are fed into the BI-LSTM module in order to train the model to learn the glosses from the video.
- Training a **Transformer** using the output from the LSTM in order to derive a complete sentence from the glosses identified.This part is similar to **Machine translation** problem in **Natural language Processing** domain of machine learning.

4.1.2 I3D model

The **I3D model** [**6**] is a unique architecture, which uses a combination of Convolutional Neural networks in order to capture the temporal as well as spatial dimensional feature of a video.

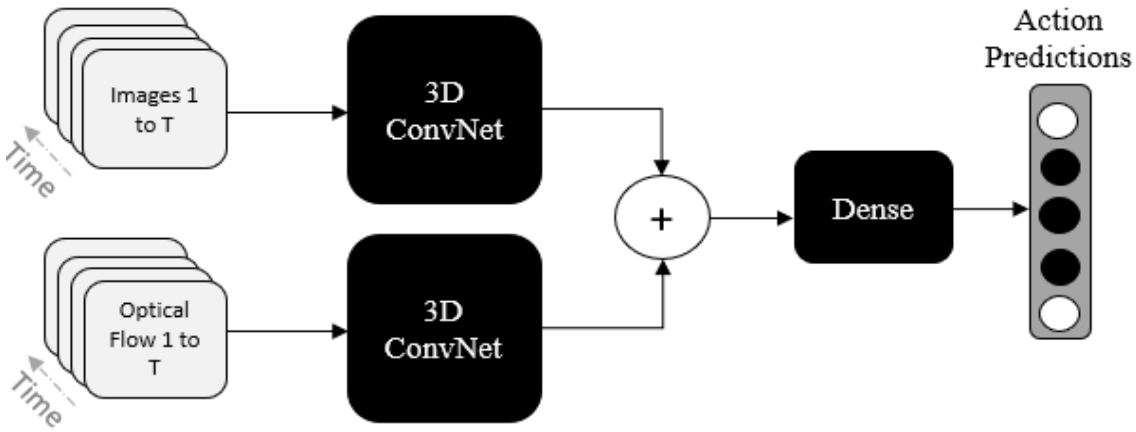

Figure 4.1: I3D architecture

In this architecture, the whole video is taken as input, frame by frame, and a 2D convolutional filter of size $N \times N$ is **inflated** to a 3D convolutional Filter of size $N \times N \times N$.

The spatial dimensional features are captures by the 3D convNet running over the frames. At the same time, a **Receptive field** is used to capture the temporal dimensional feature over the frames as an optical flow over time.

These 2 features are concatenated together and passed through a Deep Dense Neural network to identify the video's task.

4.2 Implemented Approach

We took the inspiration from the work done by **Kayo yin** [**34**] and planned the work in a similar approach where we will complete the task at hand by making several modules out of it. We planned to divide the whole project into the following Modules:-

1. Converting all videos to frames of equal length.

2. In the available literature using **STMC module[36, 34]** we can compute the spatial as well as temporal features from the frames of a video, but the problem with this approach is its computing complexity. The method takes several crops per frame and computes the feature on all of these crops, and concatenates them, which is a very complex and expensive task in terms of memory and time.

   So we propose a different approach, where we will use a pre-trained **3D Convolutional Neural network** which will help us capture these spatial as well as temporal features from all of the frames per video, with minimum space complexity. So we passed frames of a video to **Pre-trained I3D model[18, 1]** in order to capture spatial and temporal features.

3. Train following architectures to convert these features into suitable gloss:-

   - Attention Model
   - Transformer

4. **Gloss to Text**:-Since Transformers are found to be a perfect architecture for problems like Machine translation, we chose to work on Transformers for **Gloss to text** conversion.

where, **Glosses** isn't exactly a fully established sentence, in the sense that they do not make a qualified sentence in any language. These glosses are some intermediate words that are being used in a video. Using these glosses, we will make an appropriate sentence in a given language.

For example, consider we have a sentence "Harry ate an apple along with some bananas for his lunch ", then the glosses for this sentence can be "harry," "apple," "banana" and "lunch."

It is not necessary that using these glosses, a particular sentence will be formed, but the resultant sentence will surely have these glosses as a part of it.

Chapter 5

Methodologies and Implementations

5.1 Static sign language

5.1.1 Proposed models architectural implementation

With the Knowledge prevailed by various Literature we went through, we proposed and tried several models for the data set available, with various configurations. Out of which, the most successful models we've implemented (with good results) are as follows.

5.1.1.1 Deep Neural Network with Hyper Parameters tuning

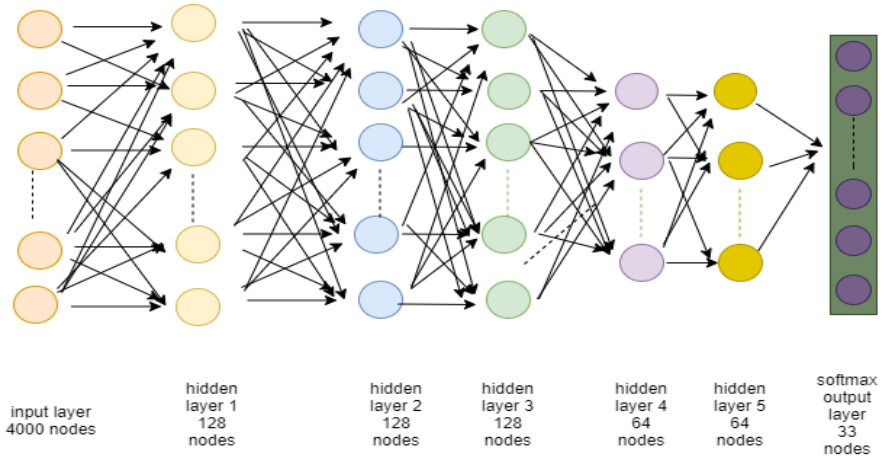

Figure 5.1: Proposed Deep Neural network

The most basic of all network we designed was a **Deep Dense network**.

Although this type of model is not suitable for image recognition due to a large number of hyperparameters, we proved that with shallow, deep models, we could achieve good results (although not comparable to CNN in the sense that the number of parameters as well as hyperparameters will increase by several folds hence time as well as space complexity will increase to a severe extent) for our project by **Hyper Parameter tuning method**.

The implemented model has the following architectural design:-

- **Number of nodes in Input layer**:-4000
- **Number of Hidden Layers** :-
  - 3 Layers with 128 nodes each.
  - Followed by 2 layers with 64 nodes each
- **Output layer**:-A dense layer with 33 nodes ,one for each output class.**Softmax Activation** is used here.

After several Hit and trial methods, the following hyperparameters gave us the best results:-

- **Activation Function**:- for our purpose we tried **sigmoid,tanh,relu and leaky relu**, and **ReLu activation** .
- **Regularizer**:- **Dropout regularizer** with value **0.2**.
- **Optimizer**:- **Adam optimizer** with β_1 **value 0.9 and** β_2 **value 0.99** .

Using this model, we trained and tested the data sets in the following manner:-

- We divided the whole data set into 2 parts training and testing
  - **Training data set**:-
    - It consists of 70% of the images.
    - For Training Purposes, we re-scaled the images in our data set into a size of 70×70 and formatted them into grayscale to reduce complexity.

    - then each item was converted into a feature matrix of size 70×70 and flattened into a vector of size 4900.

  - **Testing data set**:-
    - It consists of 30% of the images.
    - Similar to the training phase, we formatted the images into a size of 70×70 and then changed them into grayscale.
    - we converted these images into a feature vector of a size similar to training images.

we trained our model with the following approach:-

- **Forward propagation** :- the forward propagation step includes calculation of **Weights matrix and bias vector** at each layer.
- **Back-propagation and loss computation**:- after getting the outputs for the first forward pass, we calculated the accuracy. We back-propagated the weights and bias for optimization purposes, where it includes updating the weight and bias matrix for each layer until the first layer.
- **Epochs**:- we repeated the process for a total of 100 epochs.

For testing our model, we proceeded as follows:-

- images were passed similarly as in the training phase. but the weights and bias were fixed as computed after all epochs in the training phase.
- the output layer predicts some outputs.
- based on these predicted outputs and actual expected outputs evaluation metrics were calculated.
- There's no backpropagation for the testing phase.

With the above-mentioned configurations, we have achieved the best results for our data sets.

5.1.1.2 Convolutional neural network

Convolutional neural networks have sparked a revolution in the computer vision world; it would be disappointing not to go with this model.

Since the real motive of our project is to identify and recognize sign languages in real-time and just for knowledge purposes, we are testing on an image dataset; we have proposed a fundamental convolutional neural network with optimized hyperparameters.

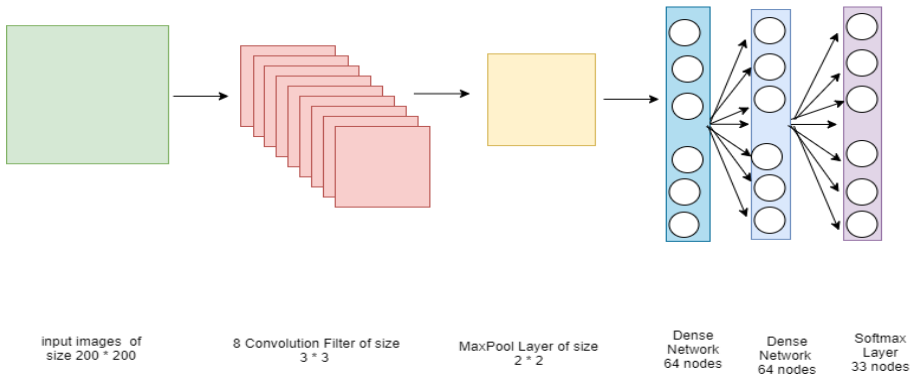

Figure 5.2: Proposed convolutional Neural network

The model has the following architectural design:-

- **Input** :- Input layer is a **matrix** of size 200×200,where each unit represent a pixel of an image normalized to a value in between **0 and 1**.
- **Filters** :- 8 Convolution Filter of size 3×3 are stacked together ,where one filter will perform convolution operation with input matrix at a time,and result will be concatenated together.
- **Pooling layer**:- A **Max Pool layer** with size 2×2 and stride of **2** is used.
- **Dense layer**:- 2 dense layer with 64 nodes each.
- **Output layer**:- a Dense layer with 33 nodes ,for each of the output class.

  following hyperparameters gave us the best results:-

  - **Activation Function**:- **ReLu activation** .
  - **Regularizer**:- **Dropout regularizer** with value **0.5**.

- **Optimizer**:- **Adam optimizer** with β_1 **value 0.9 and** β_2 **value 0.99**

  .

For using this model we did the following feature modifications:-

- The whole dataset was divided into 2 parts **Training data**(consists of 70 % of the data available) and **testing data**(consists of 30 %) of the data.
- The images were resized to a dimension of 70×70.in order to reduce computations.
- All images were RGB in nature.

Using this model, we trained and tested the data sets in the following manner:-

- **Training phase**:-
  - **Forward pass**:-the images were passed through the network in this sequence :- "$images -> convoultion\ layer -> max\ pool\ layer -> dense\ layer -> softmax\ layer$".
    where for each pass, the weights and bias for each convolution layer were calculated.
  - **Backward pass**:- the loss was computed for each pass and weight, and bias was re-calibrated based on computed loss for each layer.
  - **Epochs** :- we repeated this process for a total of 50 epochs.
- **Testing**

For testing our model, we proceeded as follows:-

- images were passed similarly as in the training phase. but the weights and bias were fixed as computed after all epochs in the training phase.
- the output layer predicts some outputs.
- based on these predicted outputs and actual expected outputs evaluation metrics were calculated.
- There's no backpropagation for the testing phase.

With the above-mentioned configurations, we have achieved the best results for our data sets.

5.2 Dynamic Sign language

For Dynamic Sign languages, the input dataset is a stream of **videos** and working with this type of dataset is a complex job. So we've proposed to break down this portion into several components, where for each component a series of architecture is implemented.

following are the breakdown components:-

- Conversion of Videos to equal number of Frames.
- Feature capturing from all of the frames for a video.
- Video (Frames) to **GLOSS** conversion.
- **GLOSS** to **TEXT** conversion.

5.2.1 Proposed models architectural implementation

5.2.1.1 Videos to Frames

Since it is not guaranteed that all the videos in a dataset will be of equal length, so it is really important to make the dataset more symmetrical so that the model can work on it.

We've converted all the videos into a series of frames using **openCV** library. The maximum number of frames in a video is 495, and the minimum was found to be 56.

To manage an equal number of frames per video, the following strategies are proposed:-

- making all the frames equal to the maximum number of frames, i.e., 495, by repeating the last frames till 495.
- making frames equal to the average of the total frames in all videos.

- Since there's hardly a change in 5 consecutive frames, therefore, we proposed to skip every 5 frames and taking the next frames into account, and taking a maximum of 128 frames per video. In case of any video falls short of frames following strategies were used:-
  - repeat the last frame till 128^{th} frames.
  - append 0 matrix till 128^{th} frames .

5.2.1.2 Frames to Features

We are using a **Pre-trained I3D[18, 1]** model from TensorFlow-hub in order to convert our sequence of frames per video into a feature vector of **512 dimensions** each.

To make frames suitable for the **pretrained I3D** model, we reshaped all the frames into a size of 224×256.

As per our knowledge, this is the first time someone has used a pre-trained model for feature extraction from video for **Sign language detection task**.

The **I3d** model is pre-trained on **Action Recognition task** and converts a video into a 4098 dimensional vector. For simplicity reasons, we have reduced this 4098-dimensional vector to a 512-dimensional vector, representing both spatial and temporal dimensional features of a video.

We Trained the model with the following approach:-

- We froze all the precalculated **Weights and bias** of the model.
- The Pretrained I3D model we used for our project returns the following things:-
  1. Feature vector embedding of dimension 512×1 for each video(frames of a video).
  2. text embedding of dimension 512×1, where the text embedding describes the action in the video.

Since the text embedding is not useful for us, so we removed that part of the model. The model took over **100 hours of continuous run** to convert all the videos to feature.

5.2.1.3 Features to GLoss (Sign to gloss)

We are proposing 2 Different architectures for this purpose. The reason being each of them is unique and has been proved excellent for this task.

Following are the proposed architectures:-

- **Attention model**

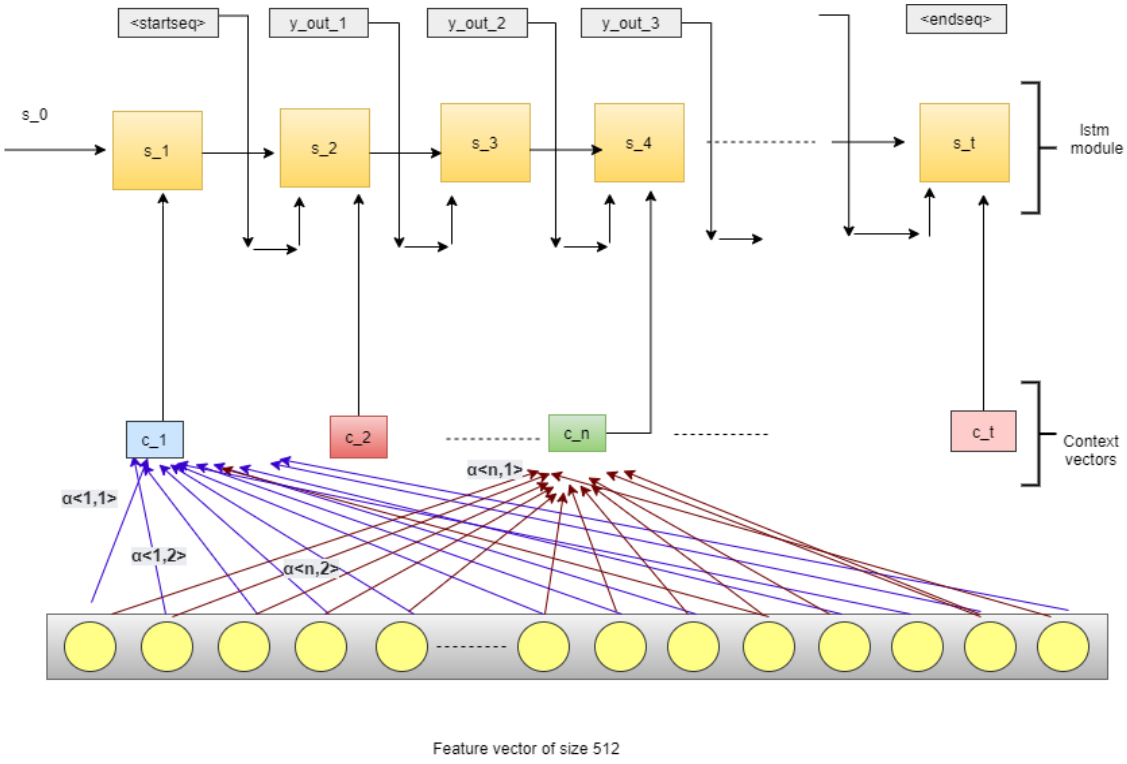

Figure 5.3: Proposed Attention network

Since we already captured the features from the video, so instead of working with the **Attention model in literature** we proposed a new architecture, where we've modified the **Encoder layer**, instead of having a series of **LSTM units** we are just passing the feature vectors.

The model has the following phase:-

1. **Training phase**:
   - **Encoder** :-During training phase,for each of the 512 feature per video is used as encoder part.
   - **Attention Layer** :- an attention weights $\alpha^{<t,t^{'}>}$ is calculated based on the features passed from the encoder layer.

- using this attention weights $\alpha^{<t,t^{'}>}$ along with the previous time stamp output,as discussed in chapter 2,**context vector is calculated**.
- **Decoder layer** :-using these context vector and previous actual previous time stamp output $y^{<t-1>}$ as input to the next time stamp t ,we are predicting the output $\hat{y}^{<t>}$.
- **EPOCHS**;- we trained the model for 30 epochs. which took over 3 hours of training time.

2. **Testing Phase**:-
   - **Encoder and Attention layers** works similar to that in training phase.
   - **Decoder layer** :- the input to decoder layer is **Context vector** and output of previous time stamp $\boldsymbol{t}$ ($\hat{y}^{<t-1>}$).

- **Transformer model**

The transformer is implemented using the following components:-

1. **Input to Encoder** :- the input has 512 dimensional vector from I3D network.

   But since Transformer works best with positional embedding, therefore we modified our input feature from a vector of 512 dimensions to a matrix of size (512×32) by using a custom function that takes a feature at the position I and does the following computation:-
   - Transform a 512-D vector into 512×32 matrix using a Deep Dense layer.
   - Multiply each 1×32 matrix with $\sqrt{i}$ where i is the position of feature ranging between (1 and 512).
2. **Encoder layer**:- cotains following components:-
   - **Multihead attention**
   - **Normalization layer**
   - **Feed Forward Layer**

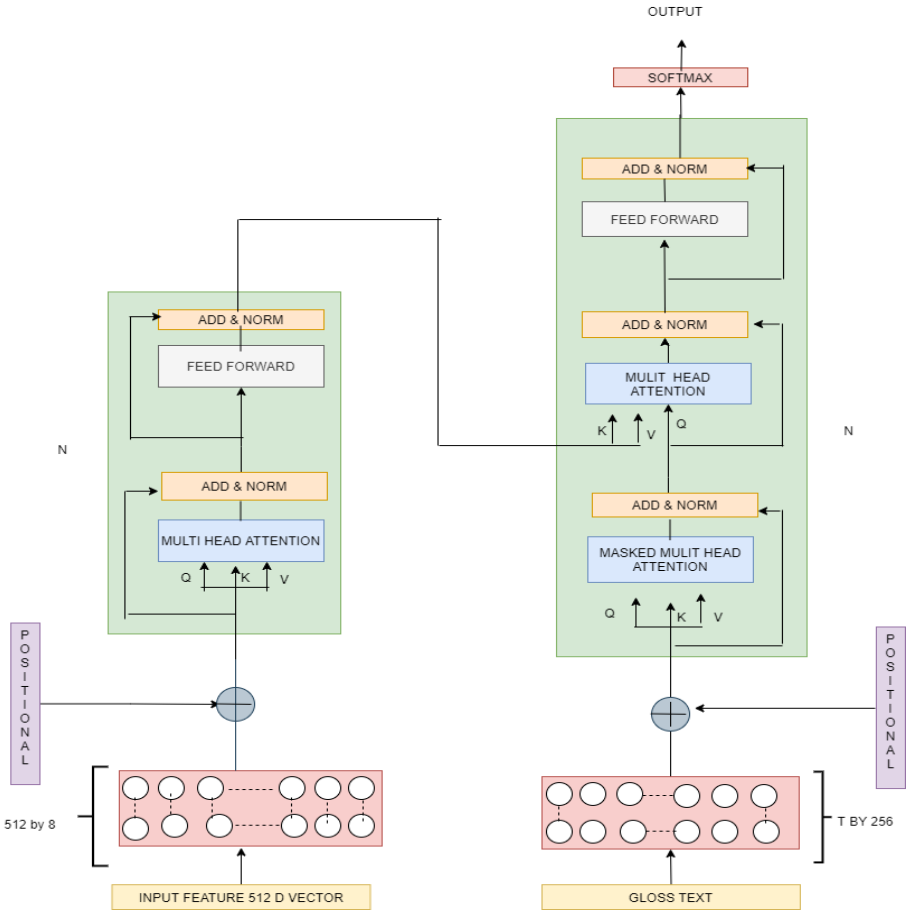

Figure 5.4: Proposed transformer for Sign to gloss

3. **Decoder layer**:-the decoder consists of the following components.
   - **Input to Decoder layer** :- it takes German language embedding for GLOSSES.
   - **Multimask attention**
   - **Nomalized layer**
   - **Multihead attention** takes **query** from multimask attention layer,and **key,value** from Encoder part.

The training and testing of the transformer model for this purpose was done as follows:-

- **Training Phase**:-

  Encoder layer

  - the feature embedding of images of 512×32 were taken as input for

the encoder part.

- we calculated the positional encoding for each time-stamp of 512 features and added them with the feature vectors.
- 3 separate matrix **Q, K, V** were calculated, which were used to calculate the multi-head attention values.
- In order to reduce overfitting, we added **Layer normalization** along with a **Residual connection** so that it can retain features for a longer time duration (to minimize the effect of exploding and diminishing weights).
- A fully connected dense layer was used to manage the size of the output vector.
- Again, a layer of **Normalization and Residual component were added**.
- This whole process was done for $\mathbf{N = 4}$ times, and finally, the results (Attention value matrix) was passed to the decoder layer

Decoder layer:-

- For the Decoder layer, the **First** input was the GLOSS embedding matrix for the first timestamp.
- for the transformer to learn better the next time stamp features are hidden by a **masked layer** so that the transformer will find it tough to train and eventually will learn better weights.
- this matrix along with the positional encoding was used to calculate the **Masked multi-head attention**.
- we passed this matrix through a **Layer norm** along with **Residual component**.
- this matrix will give us our **Query matrix Q**.
- The other input to Decoder was **output of encoder**, .which will give us our **Key (k) and Value (V)** matrix.
- we will calculate multi-head attention using these 3 matrices (Q, K, V).

- the attention values vector will be passed through normalization and residual component again.
- Finally, a dense layer followed by normalization and a residual component.
- this process will be repeated for $\mathbf{N = 4}$ times.
- the Output given by this decoder layer will include a feature embedding for the current timestamp GLOSS output.
- for next time step the actual GLOSS feature embedding matrix for next time step will be used as input to the decoder.

We trained this network for a total of 300 epochs.

- **Testing phase**:- For the testing phase, everything there will be the same except the fact that the output from the decoder at a particular timestamp will be the input to the decoder for the next timestamp.

5.2.1.4 GLOSS to TEXT

we are using a **Transformer architecture** similar to the one described in **"Attention is all you need"**.

Following are the architectural details:-

1. **Input layer** :- The input layer consists of Embeddings of GLOSS text generated by SIGN to GLOSS model.
2. **Encoder layer**:- contains following components:-
   - **Multi-head attention**
   - **Normalization layer**
   - **Feed Forward Layer**
3. **Decoder layer**:- the decoder consists of the following components.
   - **Input** :- it takes German language embedding ,with all the embedding values except current time stamp value **masked**.
   - **Multi-mask attention**

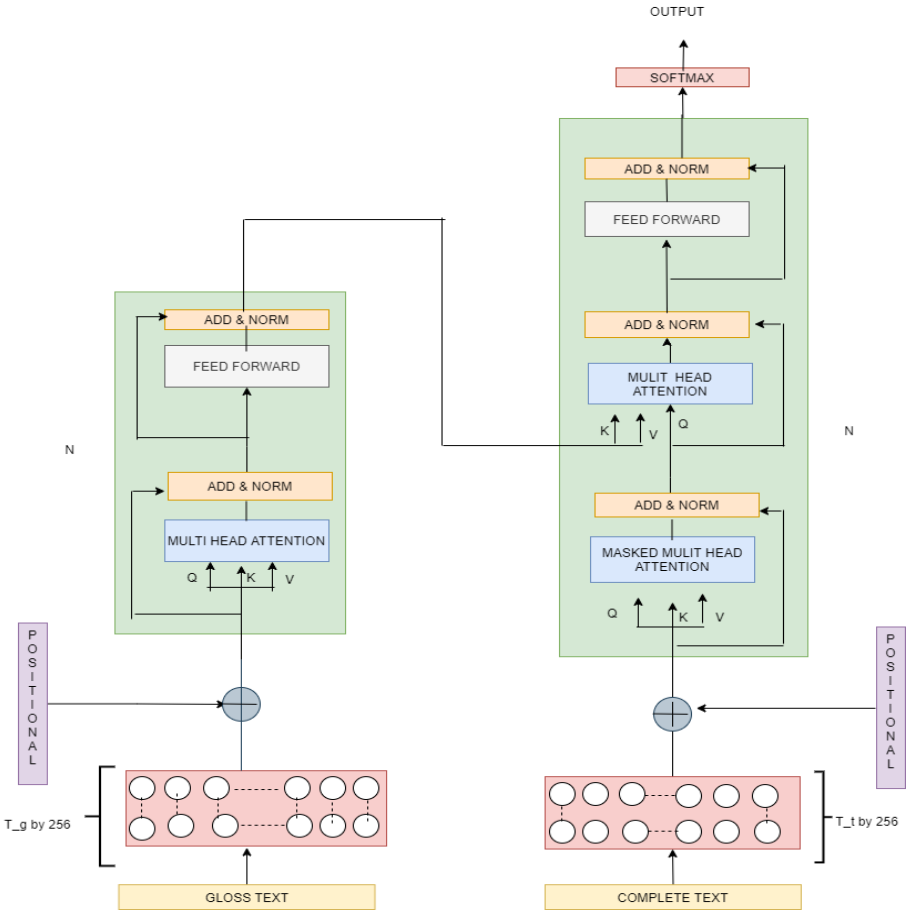

Figure 5.5: Proposed transformer for gloss to text

- **Normalized layer**
- **Multi-head attention** :- takes **query** from multi-mask attention layer,and **key,value** from Encoder part.

The **Training and Testing** will be similar to that of **Sign to GLOSS model** by using the Transformer model.

Chapter 6

Results and Discussions

6.1 Experimental setups

This section describes all the experimentation details we have used while implementing the proposed model. We will discuss the following things for all the Proposed models:-

- **Hardware details**
- **Software and packages details**
- **Dataset Details**
- **Parameters details**
- **Optimization techniques**
- **Objective function used**
- **Epochs**

6.1.1 Hardware Details

For both Static and Dynamic Models, we used both Local machines and Google Colaboratory. Following are the details:-

	Processor	**RAM size**	**GPU**	**Memory**
Local machine	Intel i5,(8^{th} gen)	8GB,ddr4	nvidia Geforce gtx 1050ti	1tb overall
google colab	-	12 GB	Tesla T4	60gb

Table 6.1: Hardware Details

6.1.2 Software and package details

Following are the Software details we have used in our project:-

- Anaconda Environment.
- Jupyter notebook.
- Python 3.7.1

Following are the packages used:-

- **Numpy**:- numpy is a scientific computational package for python.
- **Pandas** :- Pandas is a data management and visualizing package for python.
- **Tensorflow** :- Tensorflow is a Deep Leaning library
- **Tensorflow**.keras :- keras is a top layer implementation in tensorflow ,where it provides pre implemented Layers,optimizers etc for deep learning models.
- **Pytorch** :- pytorch is a deep learning library.
- **Spacy** :- Spacy is a natural language processing package.We tested this package to implement our text embedding matrix.
- **OpenCV** :- opencv is a image processing package.
- **NLtK** :- nltk is a natural language processing library.We have used **BLEU score** computation from this package
- **jiwer** :- jiwer is a nlp package,from which we have implemeted **Word error rate** computations.

6.1.3 Dataset Details

6.1.3.1 Static sign language

1. **American English Alphabets**
    - Images of 26 alphabets of the English language were used.
    - Each image has a separate folder containing 100 images each.
    - Each image has a dimension of 200 × 200 pixels.
    - Each image was an RGB image.
2. **Numeric Sign Dataset**
    - Images of 10 numbers (0 - 9) were used.
    - Each image has a separate folder containing 1500 images each
    - Each image has a dimension of 100 × 70 pixels.
    - Each image was grayscale in nature.
3. **Combined English and Numeric Dataset**
    - 23 English alphabets and 10 numeric data were used.
    - Few alphabets were removed either due to them being dynamic in nature or having similar hand signs as that of numerical data
    - example alphabet **v** and number **2** has same sign.
    - Each image was of 200 × 200 pixels and RGB in nature

6.1.3.2 Dynamic sign language

1. **RWTH-PHOENIX-Weather 2014 T: Parallel Corpus of Sign Language Video, Gloss and Translation**:-

    This dataset[29] contains the weather forecast news in German sign language over a period of 3 years (2009-2011). This dataset contains over 8000 videos performed in a controlled environment at the rate of 25fps, and the size of each frame is 210 × 260.

    This dataset contains the following things:-

- **Videos of each news forecast**.
- **Gloss per video** :-

  As discussed in **chapter 4 Glosses** aren't exactly a fully established sentence, but they just denote some of the important words that appeared in the signs and using these glosses, we will generate one of the (if not exactly) sentences that will convey the meaning of signs performed into a more understandable textual format.

- **Complete German translation of sign videos to the German text**.

We converted all the videos to frames. We also used a batch of size 54 for training purposes.

6.1.4 Dynamic dataset modification

for making this dataset suitable, we have done some modifications to it, which are as follows:-

1. **video to frames to feature**:- As described earlier in **section 5.2.1.1** and **section 5.2.1.2**, we have converted all of the videos to frames using Image processing packages and then frames to features using **I3D** model.

2. **Gloss and german text modification** :- The problem with machines is that they can not directly understand texts as we do. In order to make these text into numerical form, we made the following changes:-

   - **Embedding matrix using Spacy package** :- this package contains several languages, including German also. It contains a dictionary of over 5000 words per language, which are used to find similarities between each of the words (**Word embedding**). The embedding feature will be a vector of size (300×1).Using these embedding, we can train our model to predict the embedding vector for the predicting word.

   The problem with this method was, our Gloss data now is represented by a matrix of size $(7096 \times 35 \times 35 \times 300)$ where,

- 7096 is the total sentences in our datasets.
- 35 is the max length of sentence in the gloss dataset.
- for each input word we had, we masked it with 0 for a total length of max sentence length.
- 300 is the embedding size per word.

And the resultant matrix was of size 8 gigabytes; loading it for training purposes was not possible due to **Out of Memory error.**

We solved this problem by making our own vocabulary, which is as follows:-

- **Dictionary creation**:- We calculated all the unique words available in our dataset, including both GLOSS and German texts. This results in a vocab size of 3606 words in our Dictionary.
- **Indexing each word in dictionary**;- we indexed each word available in our dictionary based on their position in the dictionary.
- **Embedding**:- unlike in Spacy embedding, where the size of embedding was 300×1, we created a sparse embedding of size 1×1 where the number represents only the index value of a word.

  So the output of each timestamp by our models was just a number representing the index value of the word. This way, we reduced the size of the embedding matrix of all the sentences in the dataset by a factor of 300, which is as follows:-

  - **GLOSS embedding** :-$(7096 \times 35 \times 35 \times 1)$.
    where,
    - 7096 is the total sentences in the dataset
    - 35×35 max words per gloss sentence, masked to the length of 35.
    - 1 is the embedding size.
  - **German embedding** :- $(7096 \times 56 \times 56 \times 1)$.
    where,
    - 7096 is the total sentences in the dataset

* 56×56 max words per gloss sentence, masked to the length of 56.
* 1 is the embedding size.

6.2 Hyperparameters used in our project

Following tables represents the hyperparameters along with their final values we have used in our project:-

1. **Static sign language models**

Model	**Activation Function**	**Learning rate**	**Loss Function**	**Optimizer**	**Objective function**
Deep Neural network	Relu	0.001	Logarithmic cross entropy loss	Adam optimizer	Testing accuracy, F1 score
Convolutional Neural Network	Relu and Softmax	0.001	Logarithmic cross entropy loss	Stochastic Gradient Descent Optimizer	F1 score

Table 6.2: Static sign language recognition Hyperparameters

2. **Dyanamic sign lanugage models**

model components		**Activation Function**	**Learning rate**	**Loss Functions**	**Optimizer**	**Objective Function**
Video to feature	**I3D**	Pretrained	-	-	-	-
Feature to GLOSS	**Attention**	tanh , Relu & Softmax	0.001 & Custom	SCC	Adam	BLEU (1, 2, 3, 4)
	Transformer	Relu & softmax	Custom	SCC	Adam	BLEU (1,2,3,4)
Gloss to text	**Transformer**	Relu & softmax	Custom	SCC	Adam	BLEU (1,2,3,4)

Table 6.3: Dynamic sign language recognition Hyperparameters

(**SCC = sparse categorical cross entropy**)

6.3 Results and comparison with previous works

6.3.1 Static sign language

For the static Sign language detection using a Deep neural network and convolutional neural network, we have achieved the following accuracy.

model	dataset used	training accuracy	testing accuracy	f1 score
Deep Nn	Asl alphabet	90.60	80.06	0.88
	Asl(num , alphabet)	96	93	0.8
cnn	Asl alphabet	91	83	0.78
	Asl(num , alphabet)	93	89	0.88
ASl[15]	-	87.47	83.29	-
Static sign language [31]	-	-	93	-

Table 6.4: Static sign language recognition models results

6.3.1.1 Static sign language result analysis

With the models we've proposed, it can be clearly seen that we are achieving comparable results w.r.t the literature available. This proves that indeed it is possible to capture the static sign language through an image using Deep learning architectures.

The reason we are getting low testing accuracy with **cnn model and asl alphabet dataset** is that the dataset contains less number of training data, which makes it a little bit harder for the model to train properly, as we get to know that these architectures work well in the availability of huge data.

6.3.2 Dynamic sign language on rwth-phoenix weather 2014-T

Model		BLEU 1	BLEU2	BLEU 3	BLEU 4
Sign to gloss	Attention	26.38	34.71	37.43	30.82
	Transformer	22.79	26.53	10.3	17.2
Gloss to text	Transformer	18.77	30.45	35.15	29.54
Camgoz [5]	S2T	44.40	31.83	24.61	20.16
	G2T	31.87	19.11	20.16	9.94
STMC [34]	G2T	45.31	33.65	26.63	22.23

Table 6.5: Dynamic sign language recognition results

6.3.2.1 Dynamic sign language result analysis

- **Sign to gloss**:- with **Attention model** we are achieving a little bit low **BLEU-1** score as compared to work available. One reason for that could be the features we acquired are from less number of frames and only a single full crop per frame, whereas in **STMC[34]** literature, they have used a huge number of frames and crops, which might have improved the score. But since **BLEU-4** is the standard for this type of data, so we need not worry about a low **BLEU-1** score.

  But with **BLEU-2** onward, we are getting good results which even surpasses the first work in this field, i.e., **camgoz [5]**. This proves that **Attention**

models are capable of capturing and learning this type of data and are useful for these types of tasks.

The reason we are getting a low score for **Transformer** models is that transformers require a huge amount of dataset, whereas the dataset we have only consisted of **7096** training data, which might be insufficient for this cause.

- **Gloss to text**:- Since we were getting great results with **Attention model s** for **Sign to gloss**, therefore we used the gloss predicted by the attention model as an input along with the actual glosses from the training dataset for the transformer. This results in high **BLEU-4** results which proves that recognizing Dynamic sign language using this architecture and translating to an understandable text format is possible.

6.4 Discussion

With the Proposed models, we were able to prove that indeed Sign Language recognition is possible, if not easy, with the help of **Deep Learning architectures**.

Even though we have given the results only for the models which gave us satisfactory results, it is still noteworthy that we witnessed some important results and relations throughout the journey of this project, and we would like to discuss those relations in details in the subsections followed.

6.4.1 Accuracy and loss functions vs Epochs relation

Even though it is expected that with increase in the number of **Epochs** while training a model,the accuracy should ideally improve whereas the loss function should be reduced to minimal.But for some initial models we worked on, it was just not the case.We will discuss some of the worst performances along with the best results here.

6.4.1.1 Worst models (Deep Dense Neural network) Accuracy,loss function vs epochs relations

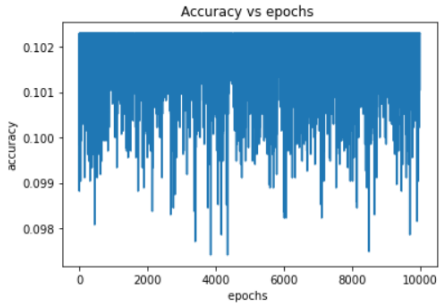

Figure 6.1: Accuracy vs Epochs

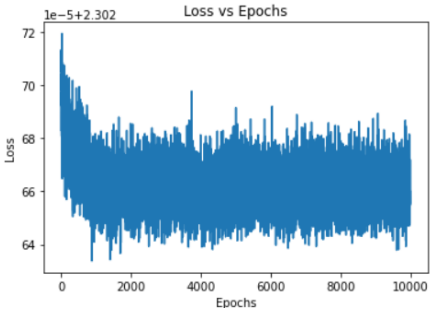

Figure 6.2: Loss function vs Epochs

6.4.1.2 Worst Result by Deep Dense Neural network

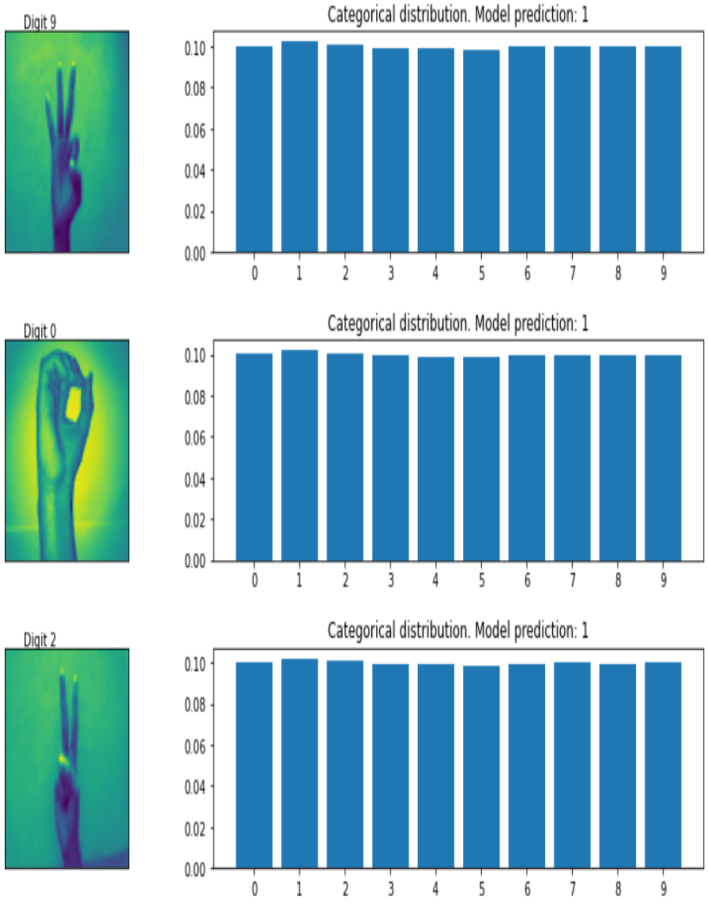

Figure 6.3: Wrong classification by Deep Dense Neural networks

The first few models designed gave us unsatisfactory results,mainly due to the following reasons:-

- Less numbers of hidden layers and hidden nodes.
- No Normalization in the input layers.
- No regularization.

it can be seen in **fig** 6.1 that the accuracy starts from as low as 9 percent,even with **Ten thousand epochs** the accuracy haven't been improved.

The **fig** 6.2 depicts that even with increase in epochs the loss is almost constant with a lot of variations in it, thus giving a very coarse graph.

In **fig** 6.3 it can be seen that due to poor accuracy, the result in predicting the Signs are Mostly incorrect.

6.4.1.3 Average models (Deep Dense Neural network) Accuracy,loss function vs epochs relations

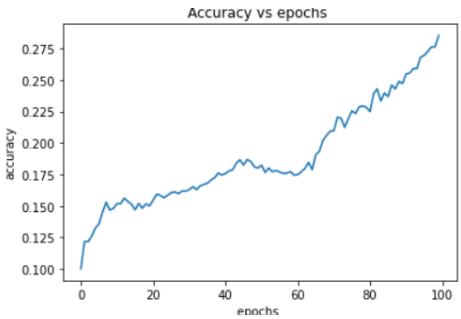

Figure 6.4: Accuracy vs Epochs

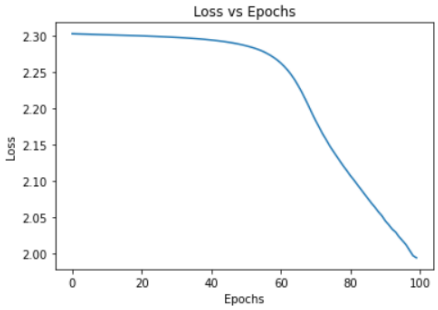

Figure 6.5: Loss function vs Epochs

With improving the models by various methods including **Normalization, good regularization etc** the quality of results improved significantly as compared to those discussed in previous section.

In **fig** 6.4 with increase in epoch the accuracy also improves. Even though the curve is not smooth yet, but it sure is showing some better progress.

fig 6.5 shows that during the initial epochs the loss was quite high, and was at a plateau region, but with higher epochs it was successful to reduce the loss to greater extent.

6.4.1.4 Average Result by Deep Dense Neural network

With the average model, the results were not the best it we can say that it improved to a significant amount. In **fig** 6.6 we can see that for some signs the model is

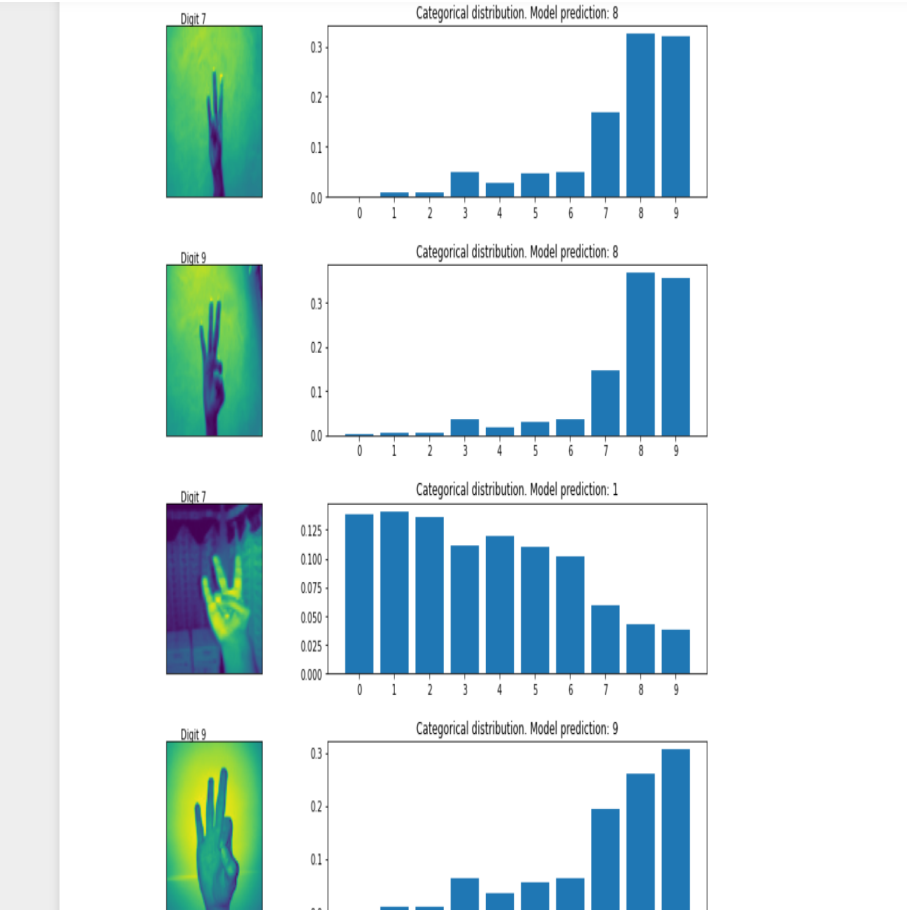

Figure 6.6: Average classification by Deep Dense Neural networks

able to identify them almost certainly with some error, whereas for some complex signs the model is not identifying them properly.

6.4.1.5 Best models (Deep Dense Neural network) Accuracy,loss function vs epochs relations

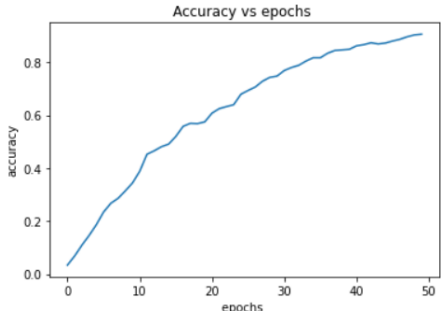

Figure 6.7: Accuracy vs Epochs

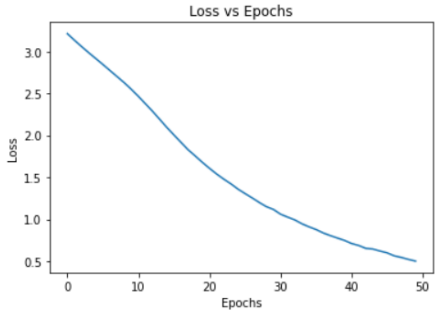

Figure 6.8: Loss function vs Epochs

In **fig** 6.7 and **fig** 6.8 we can see that with increase in number of epochs the accuracy improves significantly, whereas the loss also reduced to minimal. We can clearly see that the curve is quite smooth which shows that the model was almost certain in identifying the Signs.

6.4.1.6 Best Result by Deep Dense Neural network

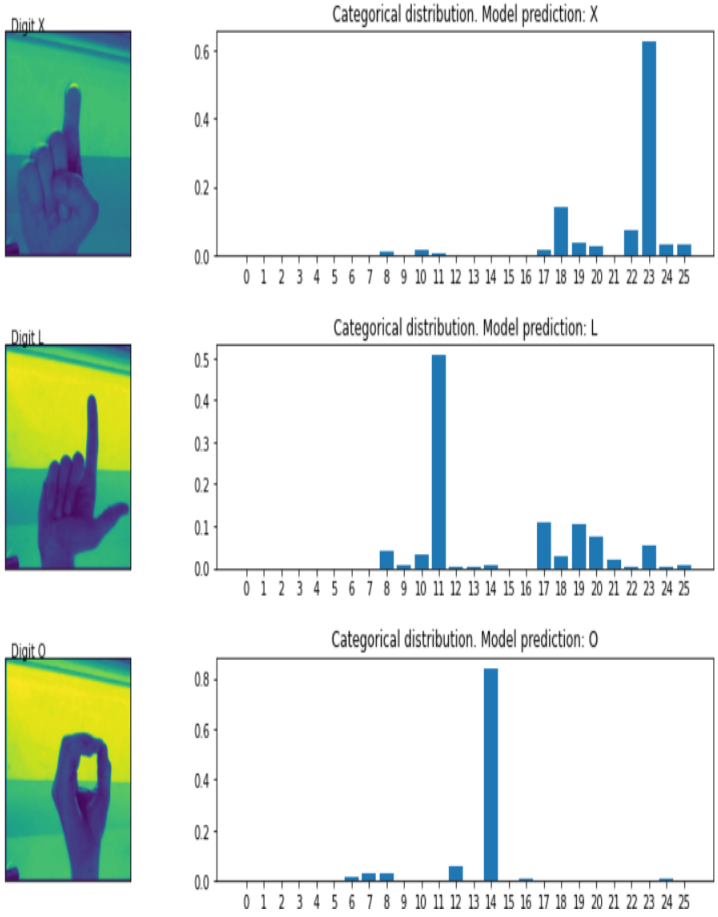

Figure 6.9: Best classifications by Deep Dense Neural networks

In **fig** 6.9 we can see how our model is identifying the Signs, and that too with minimal or no errors.

6.4.1.7 Average models (Convolutional Neural network) Accuracy,loss function vs epochs relations

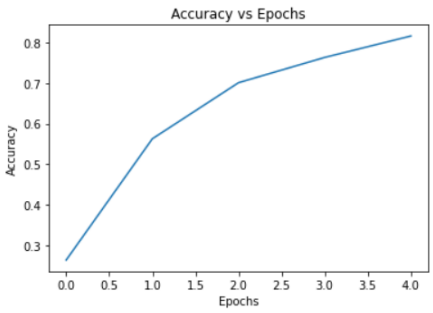

Figure 6.10: Accuracy vs Epochs

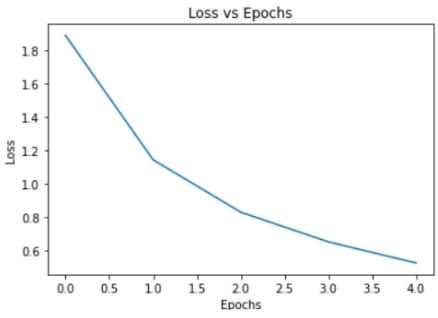

Figure 6.11: Loss function vs Epochs

Since Convolutional neural networks are far superior than Deep Neural Networks, it was quite expected that even the basic models might surpass the average Deep neural models.In **fig** 6.10 and **fig** 6.11 it can be seen that with increase in epochs the accuracy increase and loss function comes to minimal. The curves are quite smooth which shows that the models were identifying the signs without much problems.

6.4.1.8 Average Result by Convolutional Neural network

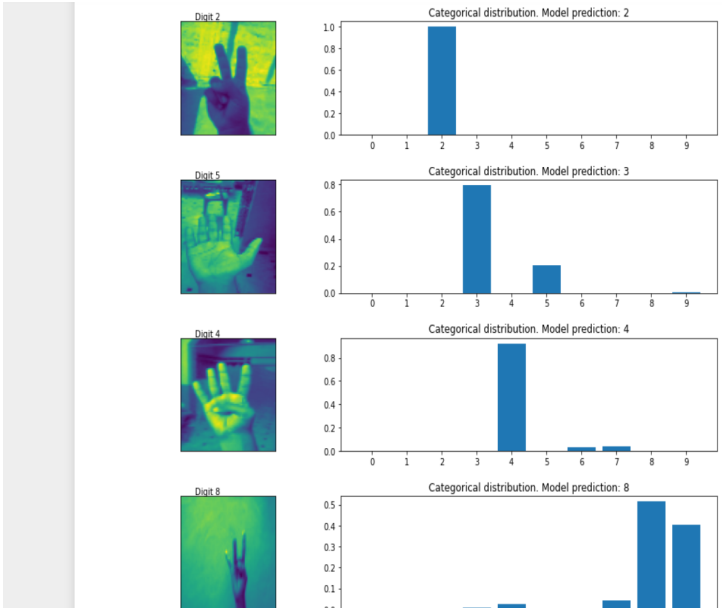

Figure 6.12: Average classification by Deep Dense Neural networks

With some initial Convolutional neural networks we were able to achieve quite astonishing results as depicted in **fig** 6.12

6.4.1.9 Best models (Convolutional Neural network) Accuracy,loss function vs epochs relations

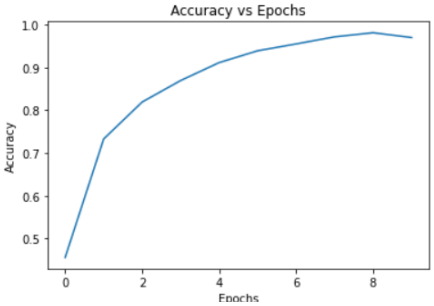

Figure 6.13: Accuracy vs Epochs

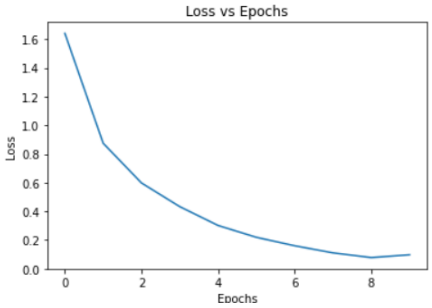

Figure 6.14: Loss function vs Epochs

With the best Convolutional neural network models we proposed the accuracy surpassed our previous Convolutional neural networks. In **fig** 6.13 we can see that the curve is quite smooth and is almost reaching to maximum value.The loss function as represented in **fig** 6.14 is almost hitting the minimum value.

6.4.1.10 Best Result by Convolutional Neural network

with the best Convolutional neural networks we proposed we were able to almost certainly identifying the signs including complex signs with almost or no error at all **fig** 6.15.

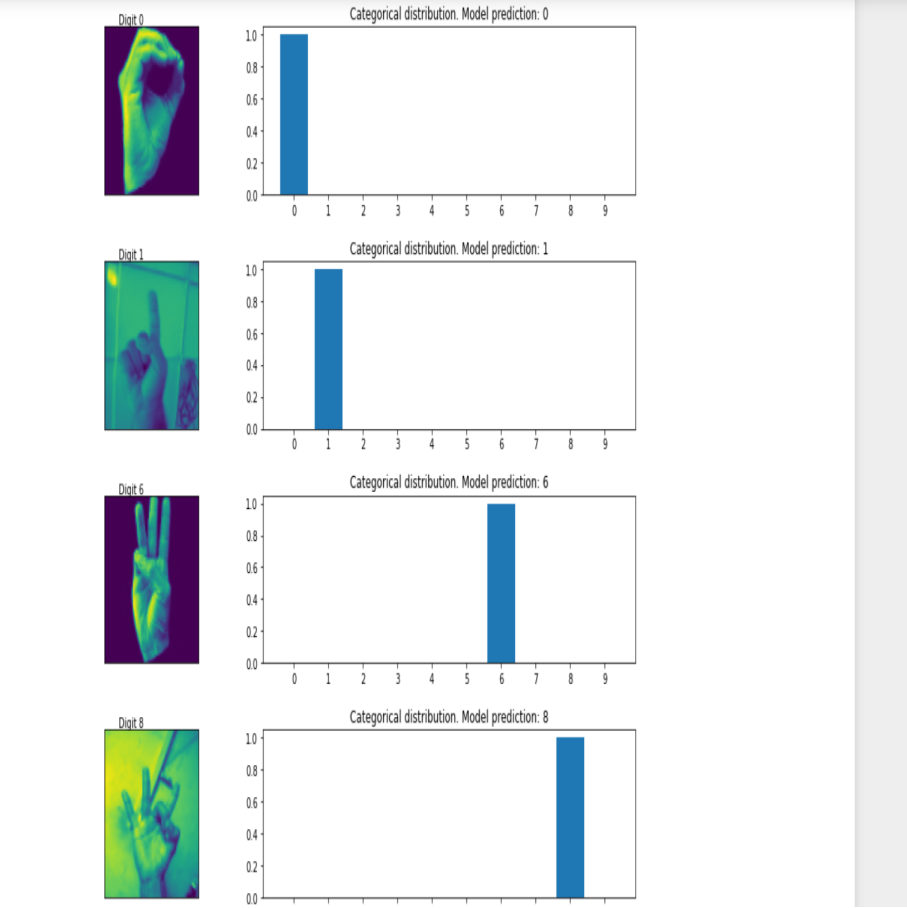

Figure 6.15: Average classification by Deep Dense Neural networks

6.4.2 Precision,Recalls and F-1 Scores

6.4.2.1 Precision,Recall and F-1 score for Deep Dense Neural networks

	precision	recall	f1-score	support
0	1.00	1.00	1.00	36
1	1.00	1.00	1.00	28
2	1.00	1.00	1.00	33
3	1.00	1.00	1.00	31
4	1.00	1.00	1.00	24
5	1.00	1.00	1.00	26
6	1.00	1.00	1.00	34
7	0.79	1.00	0.88	30
8	0.72	0.78	0.75	27
9	1.00	1.00	1.00	23
10	1.00	0.79	0.88	43
11	0.96	0.87	0.92	31
12	0.85	1.00	0.92	29
13	1.00	0.91	0.95	22
14	0.78	0.81	0.79	31
15	0.86	0.86	0.86	36
16	1.00	0.90	0.95	30
17	0.42	0.37	0.39	30
18	0.61	0.88	0.72	26
19	0.90	0.96	0.93	28
20	0.94	0.42	0.58	38
21	1.00	0.96	0.98	25
22	0.79	0.93	0.85	28
23	0.78	0.93	0.85	42
24	0.82	1.00	0.90	27
25	0.94	0.68	0.79	22
accuracy			0.88	780
macro avg	0.89	0.89	0.88	780
weighted avg	0.89	0.88	0.88	780

Figure 6.16: Precision,Recall,F1 score for dense neural network

In **fig** 6.16 we can see the precisions,recall and F-1 score for the best deep Neural network on the alphabet dataset.For some Signs the score is 1 representing it almost certainly identified the Sign, the reason can be the simiplicity in the Signature nature, where the network can easily distinguish the Sign among other signs.

6.4.2.2 Precision,Recall and F-1 score for Convolutional Neural networks

	precision	recall	f1-score
class 0	0.85	0.87	0.86
class 1	0.89	0.77	0.83
class 2	0.71	0.81	0.76
class 3	0.79	0.91	0.85
class 4	0.71	0.61	0.65
class 5	0.88	0.89	0.88
class 6	0.73	0.84	0.78
class 7	0.68	0.76	0.72
class 8	0.83	0.58	0.68
class 9	0.82	0.83	0.83
accuracy			0.79
macro avg	0.79	0.79	0.78

Figure 6.17: Precision,recall,and F1-score for convolutional neural network

In **fig** 6.17 we can see the precision, recall along with F-1 score for the best Convolutional neural network for Numerical dataset.

With a lot of models trained and tested throughout the journey of our project, we described and discussed some of the models and the result achieved by them.

And we are looking forward to making further improvements on this work with our future proposed works.

Chapter 7

Conclusion and Future scope

7.1 Conclusion

With our work, we have proved that Deep Learning methodologies are quite more than sufficient in recognizing are translating **Sign languages** into **Readable text**.

We estimated that for Sign to gloss, we can achieve better results with the Transformer model because it was the most powerful model of all the models we proposed, but it proves that it needs a significant amount of Dataset for training purposes which we were lacking. Along with that, computational power was one of the issues we faced due to the ongoing pandemic.

Using these architectures along with enormous datasets available and with high computational power, we can develop more complex and powerful models such as **BERT[9], T5[22]** which can improve the accuracy of this translated text to a lot better extent.

7.2 Future work

We were unable to test our models with high-end resources due to the ongoing pandemic. We were unable to test our models on huge data sets such as the massive **Chinese sign language dataset[35] and American sign language dataset[2]**, which together take more than 1 TB of space, so we propose to work with these datasets in the future if adequate resources are available, and we believe

that these giant datasets will be proved of much use for us. We would also like to test our work with some more advanced models available like as follows:-

- BERT
- T5

The reason being these models in recent times have created a sensation in NLP domains. And we believe that **Deep Learning** is the key to unlock these mysteries. We have to wait a bit further.

Bibliography

[1] **mil-nce/i3d**. `https://tfhub.dev/deepmind/mil-nce/i3d/1`.

[2] Vassilis Athitsos, Carol Neidle, Stan Sclaroff, Joan Nash, Alexandra Stefan, Quan Yuan, and Ashwin Thangali. **The american sign language lexicon video dataset**. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2008.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. **Neural machine translation by jointly learning to align and translate**. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. **A neural probabilistic language model**. *The journal of machine learning research*, **3**:1137–1155, 2003.

[5] Necati Cihan Camgoz, Simon Hadfield, Oscar Koller, Hermann Ney, and Richard Bowden. **Neural sign language translation**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7784–7793, 2018.

[6] Joao Carreira and Andrew Zisserman. **Quo vadis, action recognition? a new model and the kinetics dataset**. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[7] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. **On the properties of neural machine translation: Encoder-decoder approaches**. *arXiv preprint arXiv:1409.1259*, 2014.

[8] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. **Empirical evaluation of gated recurrent neural networks on sequence modeling**. *arXiv preprint arXiv:1412.3555*, 2014.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. **Bert: Pre-training of deep bidirectional transformers for language understanding**. *arXiv preprint arXiv:1810.04805*, 2018.

[10] Z Hao. **Activation Functions in Neural Networks**, 2017.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. **Deep residual learning for image recognition**. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] Sepp Hochreiter and Jürgen Schmidhuber. **Long short-term memory**. *Neural computation*, **9**(8):1735–1780, 1997.

[13] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. **The curious case of neural text degeneration**. *arXiv preprint arXiv:1904.09751*, 2019.

[14] Vladimir Iashin and Esa Rahtu. **Multi-modal dense video captioning**. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 958–959, 2020.

[15] Nikhil Kasukurthi, Brij Rokad, Shiv Bidani, and Aju Dennisan. *American Sign Language Alphabet Recognition using Deep Learning*. PhD thesis, 2019.

[16] Diederik P Kingma and Jimmy Ba. **Adam: A method for stochastic optimization**. *arXiv preprint arXiv:1412.6980*, 2014.

[17] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. **Deep learning**. *nature*, **521**(7553):436–444, 2015.

[18] Antoine Miech, Jean-Baptiste Alayrac, Lucas Smaira, Ivan Laptev, Josef Sivic, and Andrew Zisserman. **End-to-end learning of visual representations from uncurated instructional videos**.

In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9879–9889, 2020.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. **Efficient estimation of word representations in vector space**. *arXiv preprint arXiv:1301.3781*, 2013.

[20] Keiron O'Shea and Ryan Nash. **An introduction to convolutional neural networks**. *arXiv preprint arXiv:1511.08458*, 2015.

[21] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. **Bleu: a method for automatic evaluation of machine translation**. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[22] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. **Exploring the limits of transfer learning with a unified text-to-text transformer**. *arXiv preprint arXiv:1910.10683*, 2019.

[23] Herbert Robbins and Sutton Monro. **A stochastic approximation method**. *The annals of mathematical statistics*, pages 400–407, 1951.

[24] Sebastian Ruder. **An overview of gradient descent optimization algorithms**. *arXiv preprint arXiv:1609.04747*, 2016.

[25] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. **Learning internal representations by error propagation**. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[26] Christina Runkel, Stefan Dorenkamp, Hartmut Bauermeister, and Michael Moeller. *Exploiting the Logits: Joint Sign Language Recognition and Spell-Correction.* PhD thesis, 2020.

[27] Padmavathi. S, Saipreethy. M. S, and Valliammai. V. *Article: Indian Sign Language Character Recognition using Neural Networks.* PhD thesis, May 2013.

[28] Karen Simonyan and Andrew Zisserman. **Very deep convolutional networks for large-scale image recognition**. *arXiv preprint arXiv:1409.1556*, 2014.

[29] Daniel Stein, Jens Forster, Uwe Zelle, Philippe Dreuw, and Hermann Ney. **RWTH-Phoenix: Analysis of the German Sign Language Corpus**. In *4th Workshop on the Representation and Processing of Sign Languages: Corpora and Sign Language Technologies, Malta, May*, 2010.

[30] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. **Going deeper with convolutions**. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[31] Lean Karlo Tolentino, Ronnie Serfa Juan, August Thio-ac, Maria Pamahoy, Joni Forteza, and Xavier Garcia. *Static Sign Language Recognition Using Deep Learning*. PhD thesis, 12 2019.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. **Attention is all you need**. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[33] Ankita Wadhawan and Parteek Kumar. **Sign language recognition systems: A decade systematic literature review**. *Archives of Computational Methods in Engineering*, **28**(3):785–813, 2021.

[34] Kayo Yin and Jesse Read. **Better sign language translation with STMC-transformer**. *arXiv preprint arXiv:2004.00588*, 2020.

[35] Jihai Zhang, Wengang Zhou, Chao Xie, Junfu Pu, and Houqiang Li. **Chinese sign language recognition with adaptive HMM**. In *2016 IEEE international conference on multimedia and expo (ICME)*, pages 1–6. IEEE, 2016.

[36] Hao Zhou, Wengang Zhou, Yun Zhou, and Houqiang Li. **Spatial-temporal multi-cue network for continuous sign language recognition**. In *Proceedings of the AAAI Conference on Artificial Intelligence*, **34**, pages 13009–13016, 2020.