Realization and Implementation of Communication for Hearing Impaired and Inarticulate People

Department of Electrical Engineering

Muhammad Faizan Ikram	**2018-UET-NML-ELECT-27**
Tanzila Iram	**2018-UET-NML-ELECT-34**

2022

Final year project report submitted in partial fulfilment of the requirement for degree of Bachelor of Science in Electrical Engineering

Namal, Mianwali
Affiliated with
University of Engineering and Technology, Lahore

DECLARATION

The project report titled “Realization and Implementation of Communication for Hearing Impaired and Inarticulate People” is submitted in partial fulfilment of the degree of Bachelor of Science in Electrical Engineering, to the Department of Electrical Engineering at Namal Institute, Mianwali, affiliated with University of Engineering and Technology, Lahore.

It is declared that this is an original work done by the team members listed below, under the guidance of our supervisor “Dr. Sajjad Ur Rehman”. No part of this project and its report is plagiarized from anywhere, and any help taken from previous work is cited properly.

No part of the work reported here is submitted in fulfilment of requirement for any other degree/ qualification in any institute of learning.

Team Members	**University ID**	**Signatures**
Muhammad Faizan Ikram	2018-UET-NML-ELECT-27	_____
Tanzila Iram	2018-UET-NML-ELECT-34	_____

Supervisor

Dr. Sajjad Ur Rehman

Signatures with date

_____

_____

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

I acknowledge ALLAH Almighty as the most merciful and most beneficent. Without acknowledging the people who contributed to the accomplishment of this project, we would not be able to fully express our joy and satisfaction. Their constant guidance, support, and encouragement resulted in the success of our project.

We were able to finish the project with Allah's grace and the prayers of our family and teachers. We would like to thank our project supervisor, Dr. Sajjad ur Rehman, and our co-supervisors, Dr. Majid Ali and Dr. Ihsan Ullah Afridi. The experience of working with them was truly extraordinary. As a result of this project, we have acquired a great deal of knowledge that will inspire our future professional endeavors. Throughout this project, their guidance and expertise kept us on track toward success. Finally, we cannot forget the specially-abled students and teachers of Government School for Special Education Mianwali, who have been tremendously helpful in data collection and have supported us throughout the project.

Abstract

Natural language processing is one of the most growing fields of research. It deals with human-computer interaction and natural languages. Humans share ideas and thoughts with people around them using speech and hearing abilities. But this is not the case for hearing impaired and inarticulate people. Through sign recognition, communication is possible with hearing impaired and inarticulate people. This project aims to develop a mobile application-based system for recognizing sign language and converting it into text and speech, which provides smooth communication between the deaf-mute community and normal people, thereby reducing the communication barrier between them. Our proposed system will consist of a mobile application that would be used for communication. The project will mainly consist of two parts, i.e., sign-to-speech conversion and mobile application development. In sign-to-speech conversion, the sign will be recognized using a mobile camera with the help of computer vision, image processing and machine learning algorithms. The recognized sign will be converted into text, and text will be used to generate speech. With the incorporation of the mobile app, this system will ease people with disabilities to communicate with non-disabled people and reduce the communication barrier between them.

1. Introduction

Natural Language Processing (NLP) is an emerging field that helps in conveying information and meaning with semantic cues such as words, signs, or images. Sign Language is a very convenient way for hearing impaired and inarticulate people. But for other world, communication with hearing impaired and inarticulate people is difficult because a normal person cannot understand their sign language in normal circumstances. This difference in society can be minimized if we can program a machine in such a way that it translates sign language into text or speech. This project utilized computer vision and image processing techniques to develop a system that can convert sign language into textual and audio form with a mobile application interface, thereby, reducing the communication barrier between specially-abled and non-disabled people. It is a reliable alternate mode of communication that does not need a voice and is critical for individuals suffering from hearing and speech disabilities to improve their quality of life. This project aims to meet UN's SDGs 8.5, 10.2 and 10.3. It aims to increase the productive employment of all persons including persons with disabilities. The second purpose of this project is to reduce inequalities and communication barriers in society.

The sign language of hearing-impaired and inarticulate people will be recorded in the data collection phase in the form of videos consisting of words and sentences that are used in daily life and emergencies. Since video is dynamic in nature and it is a collection of images that are referred to as frames, therefore it is not much different from an image classification problem. These frames will be processed in sequence and will be used to extract features with the help of deep learning algorithms, and then classify those frames based on these extracted features.

A video consists of an ordered sequence of frames that contain special information and a whole sequence of the video contains main information. So a video classification model with a Convolutional Neural Network – Recurrent Neural Network (CNN-RNN) architecture will be trained on the existing dataset and deployed on the mobile application. This machine learning model will be used to translate the sign language of hearing impaired and inarticulate people into different words and sentences which belong to 29 different classes. This report is structured in different chapters which will describe the complete methodology and design process of the project including a literature review.

Chapter 2

2. Literature Review

2.1 Machine learning

Machine learning is the type of artificial intelligence that enables computers to predict efficiently without being explicitly programmed. The machine works in the same way as human systems work to solve problems. In machine learning, we train our machine on the dataset being used. There are different machine learning models that have been trained on different types of datasets. These pre-trained models can be used for different types of applications. We can use them to train on our dataset (of the specific domain) so that we can get the required results. These models use the past information as input and predict the output by extracting features from the past data. After training the model is evaluated and then tested on unseen data so that model can perform better on testing data as well. While training the model it should be noted that it performs better on both training and testing data. While training there comes two types of issues i.e.

- **Over Fitting**

Overfitting occurs when the model performs perfectly well on the training data and is unable to perform better on unseen data. This means that model has trained itself for the training data only and it is unable to perform well for the testing data.

- **Under Fitting**

Under fitting occurs when a model is unable to perform on both training and testing data. This is due to the uneven/noisy dataset being provided for both training and testing. The model is neither good for training nor testing. Machine learning is further divided into four subparts i.e.

1. Supervised Learning
2. Unsupervised Learning
3. Semi-Supervised Learning
4. Reinforcement Learning

2.1.1 Supervised Learning

In supervised learning models are trained on labeled datasets, they can effectively perform training on labeled datasets. In this type of dataset, the labeled dataset acts as a teacher that ensures the model learns well and performs well on both training and testing data [1].

Supervised learning is further divided into:

2.1.1.1 Classification

It is the process of categorizing the data into different classes depending on the provided dataset and the labels along with them. The categories can be made on both structured and unstructured data [2].

2.1.1.2 Regression

Regression is the predictive modelling technique used in machine learning to predict the continuous outcome depending on the relationship between input features and output. It's a relationship between independent variables (input) and dependent variables (output) [3].

2.1.2 Un Supervised Learning

In Un supervised learning the machine learning models are trained on the unlabeled dataset. The machine trains itself on the unlabeled dataset and extracts multiple features from it by itself to predict the outcome. In this type of learning the machine is intelligent enough to extract meaningful information to predict the output [4].

2.1.3 Clustering

Clustering is the type of unsupervised learning. In this, the grouping of original data is performed using unlabeled data. The grouping depends on the features being extracted by the machine itself. It is the extent of similarity between the data points and grouping them based on extracted similar information in data points [5].

2.1.4 Semi-Supervised Learning

Semi-supervised learning is the hybrid of supervised learning and unsupervised learning. In this type of learning mostly the labeled dataset is being used but the machine is allowed to train itself by exploring the meaning of information or features from the provided dataset [4].

2.1.5 Reinforcement Learning

Reinforcement learning is the trial and error reward method. In this method there exists an agent that performs different actions and interacts with the environment to perform the actions that maximize the expected reward in a given amount of time [4].

The following algorithms are used in reinforcement learning;

1. Q-Learning
2. R-Learning
3. TD-Learning

2.2 Neural Networks

Neural Networks are the method of artificial intelligence in which a machine is trained in such a way that it works in the same way as the human brain. Neural networks are tuned on the training data to improve the accuracy on time after enough training is being performed. It is a type of deep learning that uses interconnected nodes in a layered structure that is similar to the human brain. Artificial neural networks are the same as the human brain. The human brain is made of different types of neurons interconnected with each other in a complex order similarly the artificial neural network is also made up of multiple nodes connected in a complex way to solve a particular problem. There are some weights for each neuron that are updated on time to perform better training and to improve the accuracy to get the solution to the required problem [6].

2.2.1 Neural Network Architecture

2.2.1.1 Input Layer(s)

The input layer takes input from the outside world. it processes the input data and analyzes it and sends it to the next layers [6].

2.2.1.2 Hidden Layer(s)

Hidden layers are next to input layers. They take the data from the input layer(s) and process them. There are multiple hidden layers present in a neural network. After processing the data from input layers the hidden layers send the data to output layers [6].

2.2.1.3 Output Layer(s)

Output layers are next to hidden layers. If the output is binary, then the output layer consists of only one neuron that gives an output in the form of 0 or 1 or yes or no. The output layer gives the final result after processing the data from the hidden layer [6].

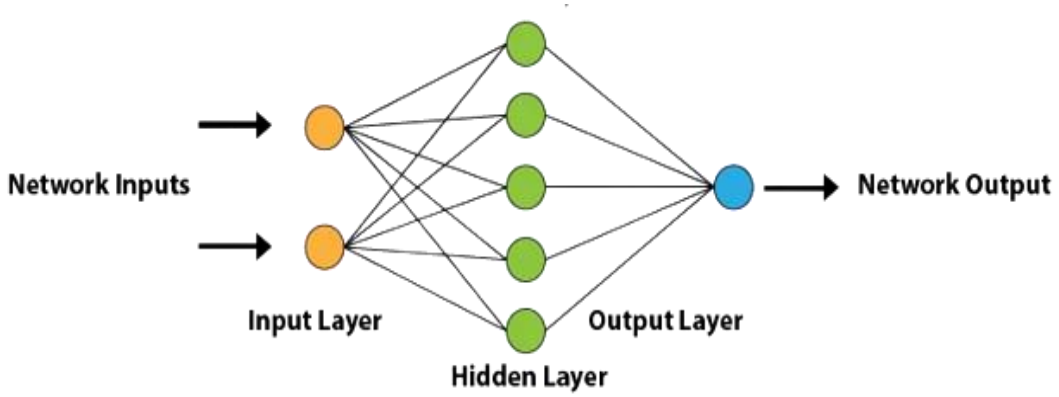

Figure 1: Neural Network Architecture [7]

Neural networks use the following algorithms to perform different operations,

1. Forward Propagation

In forward propagation, the input values are propagated from input layers to the hidden layers by multiplying the weights of the previous neuron's output with the new weights and passing them to the activation function to limit the output in a range. The process goes on for all the hidden layers and then the output of hidden layers is passed to the output layer to give the final result. The output layer also calculates the error by taking the difference between the predicted result and the expected result.

2. Backward Propagation

Backward Propagation works on the error calculated by the output layer in forward propagation. The backward propagation calculates the contribution of each weight to the output by using the derivative. These weights are then adjusted and updated for the next turn of forward propagation. This process goes on for each epoch until it results in a minimum error and the desired output is achieved.

2.3 Deep Neural Networks

A deep Neural Network is similar to the Artificial Neural Network with multiple layers in between the input layer and output layer. They are good at solving both linear and nonlinear

problems. The learning process of deep neural networks may be supervised or unsupervised depending on the dataset being provided. Deep neural networks are complex in structure due to the presence of multiple layers between the input layer and output layer. Due to their complex structure, they can solve complex problems like image classification, video classification, facial recognition etc. Because of their complex nature, they require high computational power for the training or learning process. They can extract significant features from the input data to solve complex problems.

Deep Neural Networks are of multiple types. Some of the famous types includes,

2.3.1 Convolutional Neural Network (CNNs)

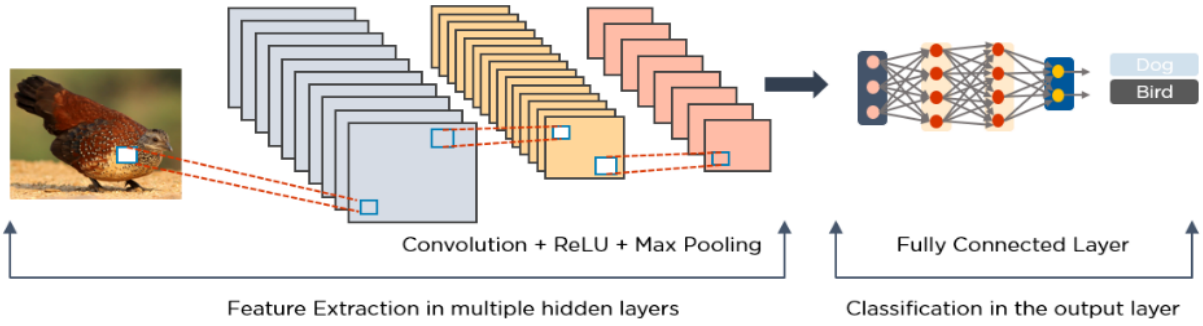

Figure 2: Convolutional Neural Network [8]

2.3.2 Recurrent Neural Network (RNNs)

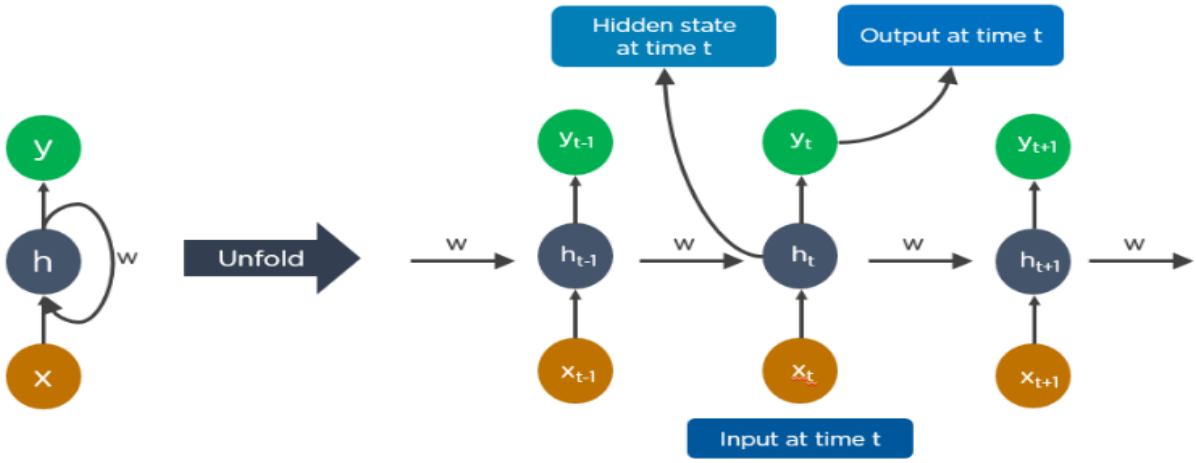

Figure 3: Recurrent Neural Network [8]

2.3.3 Multi-Layer Perceptron (MLPs)

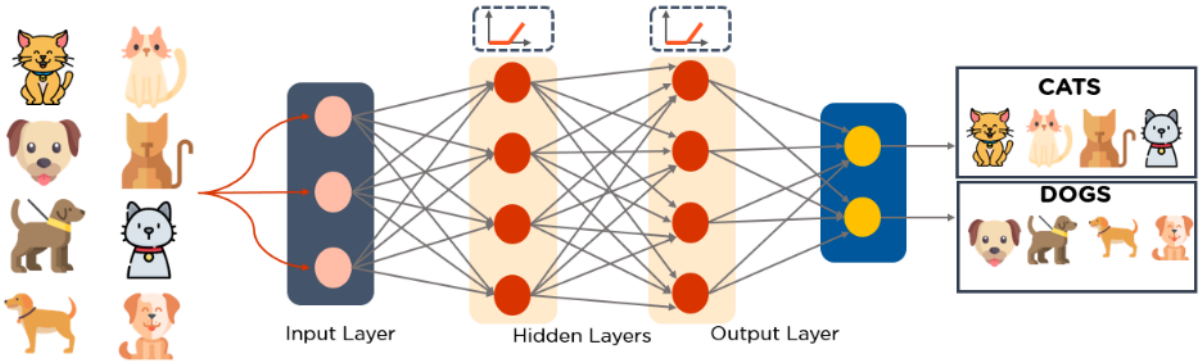

Figure 4: Multi Layer Perceptron [8]

2.3.4 Generative Adversarial Networks (GANs)

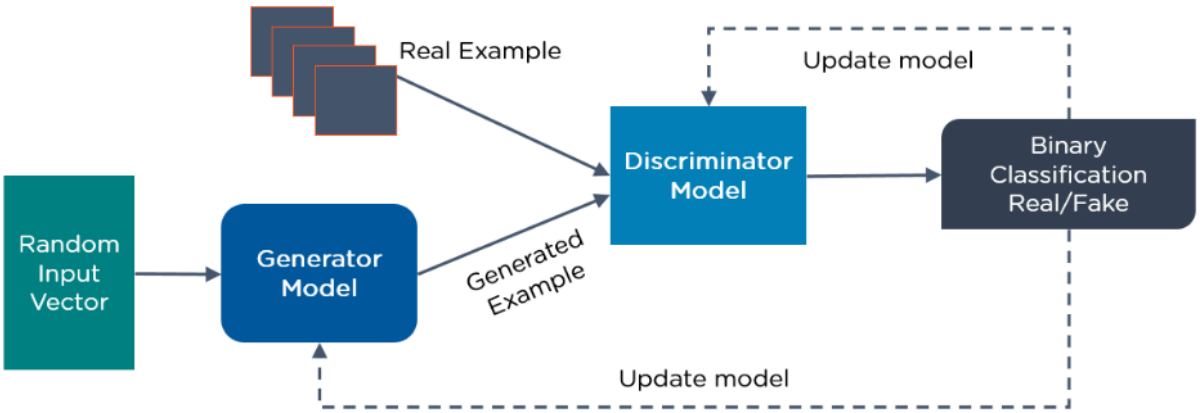

Figure 5: Generative Adversarial Networks [8]

2.4 Deep Neural Networks Applications in NLP

Natural Language Processing (NLP) is an emerging field of research. It deals with human-computer interaction and natural language. Computational algorithms are designed to analyze and represent the human language. There are a variety of deep learning models being used for natural language processing. NLP also includes sign language. Sign language recognition is an important field of research and a lot of related work has been done in this area for different fields. Sign language recognition was done by vision-based approaches, data glove-based approaches and machine learning methods like Convolutional Neural Networks, Artificial Neural Networks, Fuzzy Logic, PCA, LDA, Genetic Algorithm and SVM, etc. These techniques utilized different approaches consisting of Hand segmentation, Facial expression, Feature extraction, or Gesture recognition. Many researchers have utilized these techniques for sign language recognition. P. D. Rosero-Montalvo proposed a method of sign language

recognition based on intelligent gloves using machine learning techniques in which he used hardware-based gloves [9]. The accuracy of the system was 85% and it utilized hardware-based flex sensors to record the sign gestures which resulted in extra cost and the user have to carry additional equipment with him [9]. Mahesh Kumar NB designed a web application-based system to convert sign language into text using Linear Discriminant Analysis (LDA) approach [10]. In this system, the researcher used images of hand signs and extracted features from them to compare with the existing database to classify the sign and generate respective text. Only hand gestures were classified in this system and signs related to other body motions were ignored [10]. M. Mirzaei proposed a vision-based method using Augmented Reality (AR) and Automatic Speech Recognition (ASR) technologies that combine audio, video and facial expression of the signer to translate their sign language [11]. In this system, facial expressions of the subject were recorded and the use of AR technology helped the narrator and signer communicate with ease. But the system was expensive in terms of hardware resources and computational cost [11]. A maker-free, visual Indian Sign Language (ISL) recognition system was developed by Kanchan Dabre in which a machine learning model for sign language interpretation based on webcam images was used to extract features of hand images and classified with Haar Cascade Classifier [12]. This system also used hand gesture images only and not facial expressions or full-body motions. Also, the web application based system made it hard and non-versatile to use in daily life activities. Ankit Ojha presented Convolutional Neural Network (CNN) architecture to recognize the American Sign Language (ASL) for numbers and the alphabet [13]. The proposed method used OpenCV for real-time hand pose detection and augmentation and fed these images to the CNN model and trained it to recognize the gesture and convert them to textual content and speech. This system was developed to classify the American sign alphabets only and therefore the utility of the system was limited. A large-scale video classification model based on a spatial-temporal network and a connected multiresolution architecture was proposed by Andrej Karpathy in which a video-based approach was used to classify 1M YouTube videos [14]. Specifically, this proposed architecture was used to recognize human activities and was expensive in terms of computation and performance.

2.5 Deep Neural Networks Applications in Video Classification

Video classification is the most important field of this era. Video Classification is similar to image classification, but in video classification, there is a need to keep the sequence of frames. Multiple frames are used together in a sequence to make a full video. Deep neural networks like

CNNs extract the features from the frames in the video. In order to keep the sequence of frames RNNs are used. RNNs have the memory element to keep the record of previous frames.

In this project, we will also use vision-based approaches like image processing and machine learning to develop a sign language recognition system with an easy to use mobile application interface. We will be using video classification to extract frames of video and process them to extract meaningful features. Unlike image classification, videos cannot be easily processed due to their temporal and time-series property. Videos have a fixed-sized architecture that has both spatial and temporal information. Specifically, we will be using a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN) architecture for spatial features and temporal features respectively. This architecture consists of CNN, GRU and LSTM layers which is a hybrid architecture also known as CNN-RNN. We will also experiment with other large-scale video classification models like ConvLSTM which are also the hybrid architecture of the same kind. In the end, we will be comparing the results of our developed system with other state of the art models to evaluate the performance and efficiency.

Chapter 3

3. Methodology

For this project we have used the following process:

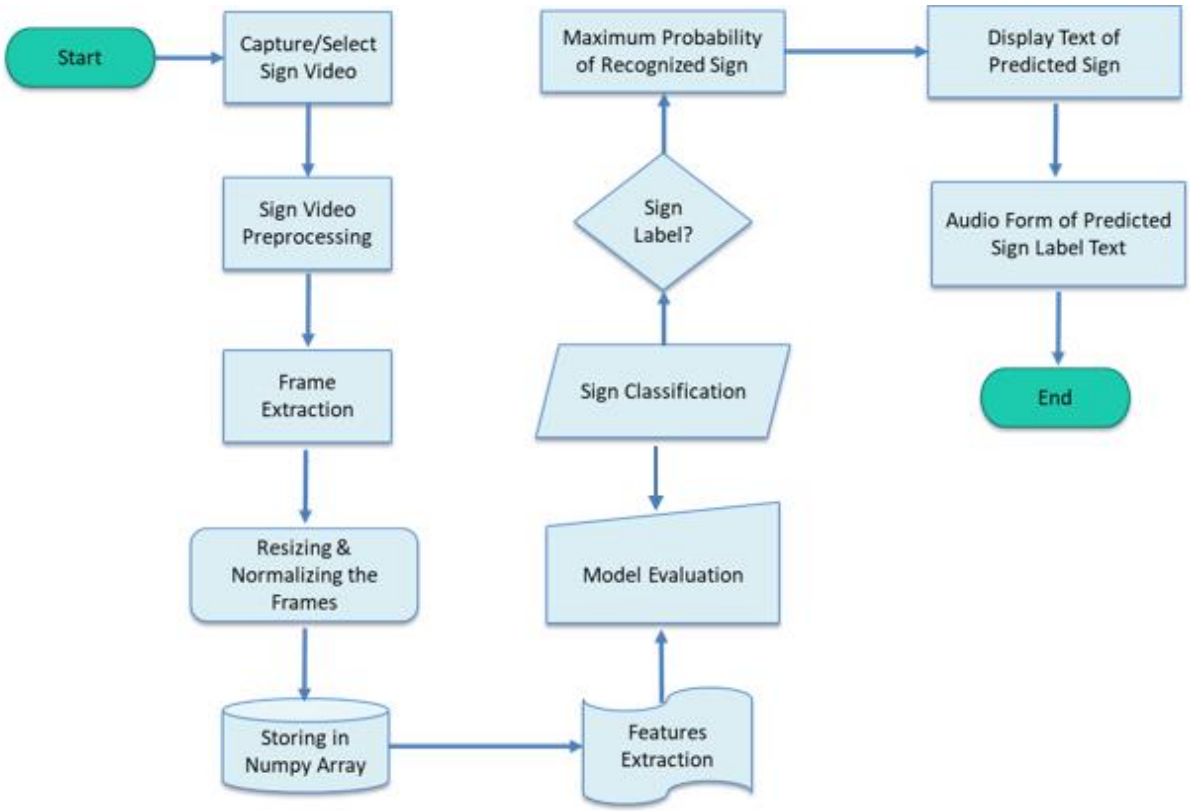

Figure 6: Methodology

3.1 Design Process

After the literature review, we moved towards the implementation of the proposed project. First, we need an ample amount of datasets that can be used for model training and processing. After we get the video dataset, the next step is to pre-process the dataset which includes background extraction, frame extraction, resizing & normalizing the frames, and then they are stored in a Numpy array. After frame extraction, we need to extract the meaningful features from these extracted frames to perform classification. Since video is a series of individual images and therefore, we may treat video classification as image classification with a total of N times, where N is the total number of frames in a video. The step-by-step process of this project is explained below

3.2 Data Collection

The first step is Data Collection. We defined the area of interest consisting of categories that we want to address in our project and train the classification model on it. These include words and sentences that are normally used in daily life activities and emergencies. The list of the words and sentences is given below:

Table 1: Dataset Recording List

Sr. No	Sentences	Sr. No	Sentences
1	Hi/Hello	**16**	Where are you going?
2	Allah Hafiz/Bye	**17**	I don't understand.
3	Yes/Alright	**18**	What is the time?
4	No	**19**	I need your help
5	Excuse me/I am sorry	**20**	Can you help me?
6	Thank you	**21**	I am looking for this
7	Who are you?	**22**	I cannot speak
8	I am a student	**23**	Clock
9	He is my brother	**24**	Door
10	What is your name?	**25**	Engineer
11	How are you?	**26**	Mechanic
12	I am good	**27**	Emergency
13	Nice to meet you	**28**	Accident
14	Leave me alone	**29**	Call the ambulance
15	I want to go to school/outside	**30**	Invalid Sentence/Video

For data collection, the local government-owned school named *Government School for Deaf and Dumb Mianwali* was approached to allow the video recording of deaf-mute students for project purposes. We made the setup and kept normal background and lighting conditions to record videos. The subjects were asked to perform signs and gestures for the tasks listed in Table 1. We recorded video samples for each sign and processed them for better use.

Along with above classes we also have to add another class i.e. "invalid class". So that if a video other than sign video is given then it can predict it as Invalid video. For this extra class there is need to collect more data in order to perform the model training. This class consists of different videos other than sign videos that are considered to be invalid. For this we collected

different types of videos including scenery and different animated videos to perform model training over them with a label of "Invalid Class". Some of these videos are collected by us that are basically the videos consisting of a scenery. We also collected animated videos dataset from YouTube and google in order to better perform the model training over the extra class. The picture of the data collection set is shown below in Figure 7:

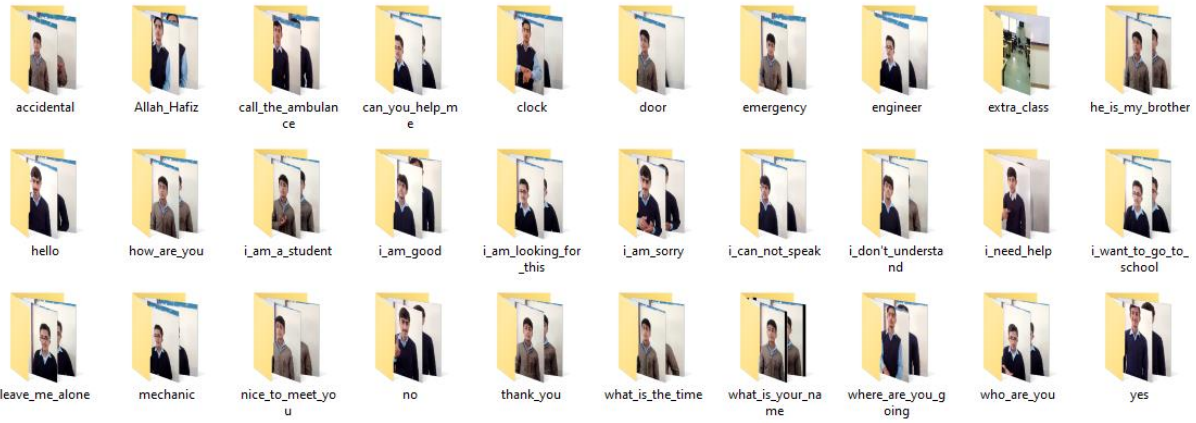

Figure 7: Folders having video sample dataset

3.2.1 Meta Data

Table 2: Metadata of recorded dataset

No. of Subjects	**30**
No. of Categories/Classes	30
No. of Samples per Category/Class	70-80 (for 10 classes)
Total no. of Samples	**1170**

3.3 Mathematical Model

We have used CNN-RNN architecture and ConvLSTM model as explained earlier. These are deep learning algorithms that are very powerful and mostly used in solving computer vision and image or video classification problems. These models are sequential and help to execute functional mathematical operations on the dataset and also maintain the long-range sequences in time-series input data. We have utilized these models for video classification as well as for maintaining the sequence and temporal information.

3.3.1 Convolutional Neural Network (CNN) Detector

CNN is a powerful class of neural networks that are inspired by the actual perception that takes place in our brain. This is highly useful in solving computer vision problems. It uses a filter/kernel to scan the features of an image and uses convolutional computations to set appropriate weights to detect and classify those specific features. The Keras application offers CNN's pre-trained models like InceptionV3, ResNet50, VGG-16 etc. to extract the features for the input dataset. These models are trained on a large amount of datasets and provide transfer learning ability in our code. The performance of these models is high as compared to the model created from scratch. So we have utilized its ore-trained models and trained extracted features in RNN algorithm.

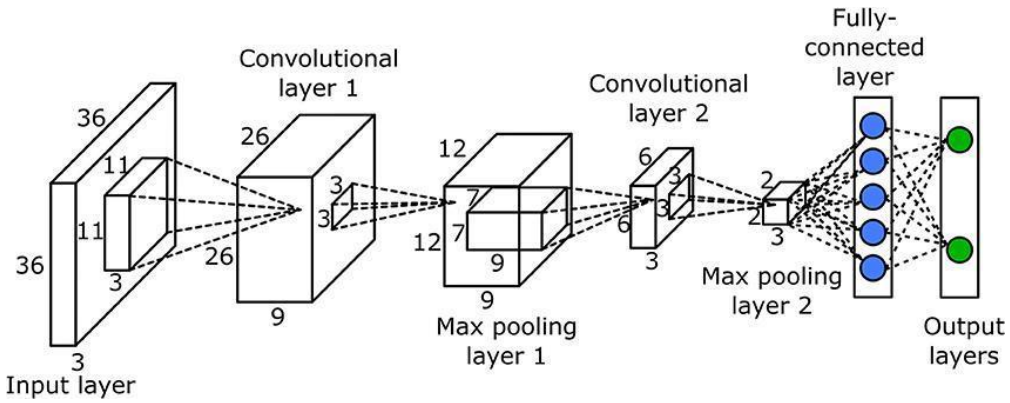

Figure 8: Convolutional layers of an image/frame deep learning model [15]

3.3.2 Recurrent Neural Network (RNN)

RNN is also a type of Neural Network where the output from the previous step is fed as the input for the current step in a layer. Because of this property, RNN is a model used in Sequence Classification or Video Classification problems in which time-series data is involved. We have used this network for temporal processing in our project.

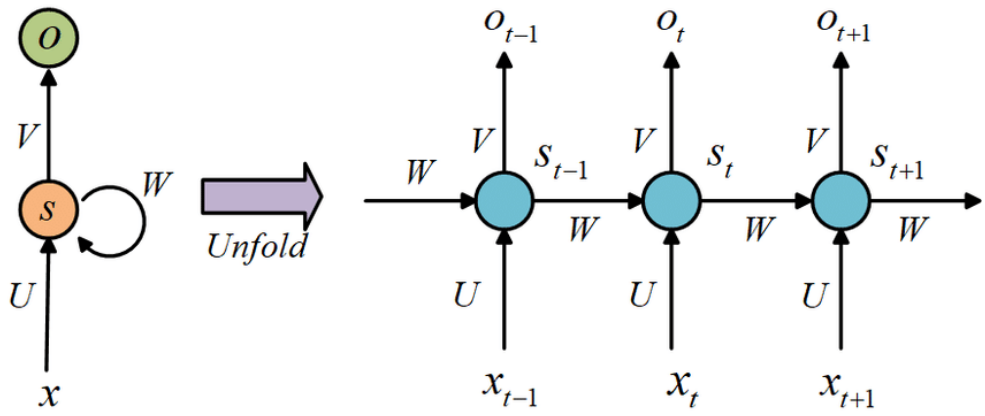

Figure 9: Basic structure of RNN [16]

RNN is very useful in predicting sequences as it accepts current and as well as previous input but the drawback of RNN is that it cannot handle long term sequences. This is called the Vanishing Gradient problem where the sequence of long-range dependencies is forgotten by the model. So, to counter this problem, we have used extended RNN architectures known as Gated Recurrent Unit (GRU) and Long Short Term Memory (LSTM). These types of RNN are more powerful in handling long term dependencies and help in handling time-series data for a long sequence of time. A detailed explanation of these architectures is given below:

3.3.3 Gated Recurrent Unit (GRU)

This is a very strong model to maintain and remember the sequence in long term decencies in the model. The GRU has different types of gates that enable it to learn the long term sequence. It has a Reset gate and Update gate in its architecture. The model accepts previous input, that is h(t-1) and current input x(t) just like RNN, but it has a special cell called the memory cell. This memory cell is updated at every time step in the model. All the inputs combine and give output and as well as generate the new value of the memory cell h(t).

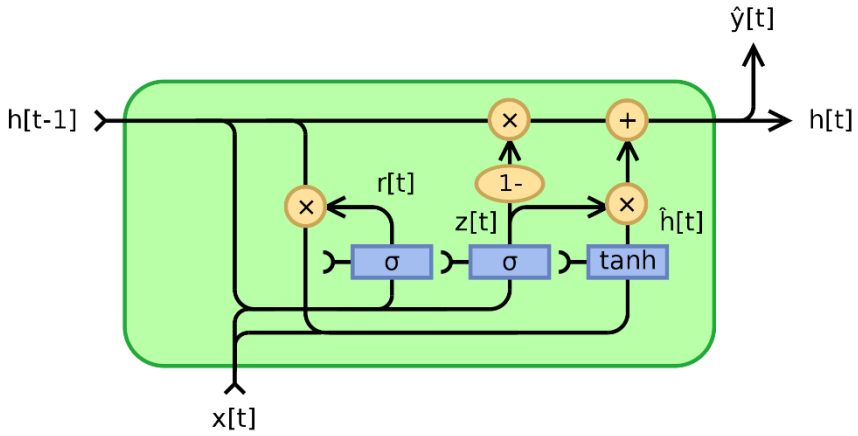

Figure 10: GRU structural building block [17]

3.3.4 Long Short Term Memory (LSTM)

LSTM is another powerful type of RNN and it modulates the flow of information just like the GRU. But the major difference is that it has a Forget Gate. This gate decides which part of the information to forget and which part of the information to retain and store for the next time step. It helps to store the most relevant information and forget the irrelevant information. Then the output is passed to sigmoid activation and update gate. The memory cell c(t) updates itself at every time step and acts as the previous input for the next time step. This architecture makes LSTM more complex in computations but most powerful at the same time, whereas, GRU's simple architecture makes it more efficient and easy to implement.

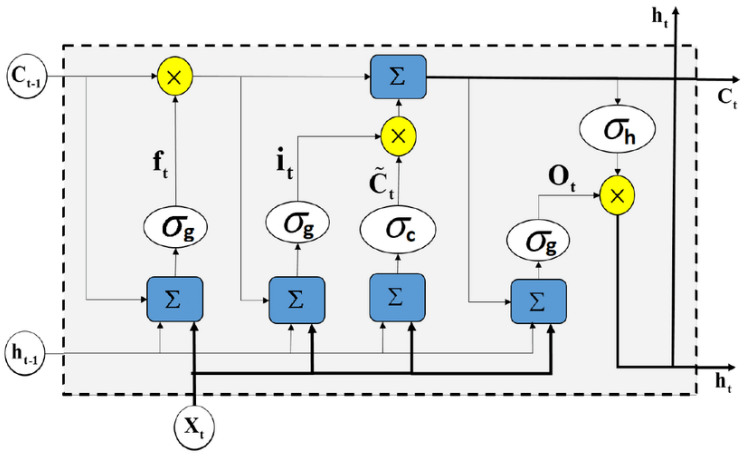

Figure 11: LSTM structural building block [18]

3.3.5 Convolutional Long Short Term Memory (LSTM)

In our project, we are dealing with spatial-temporal predictions. For this kind of dataset, we used ConvLSTM which is a type of recurrent neural network. It has a convolutional structure that determines the future state of a certain cell in the input image grid from the current input and past input states of its local neighbours. Since, we have a sequence of images in a video dataset so in this case, we have maintained the order and time-series nature of the video using ConvLSTM.

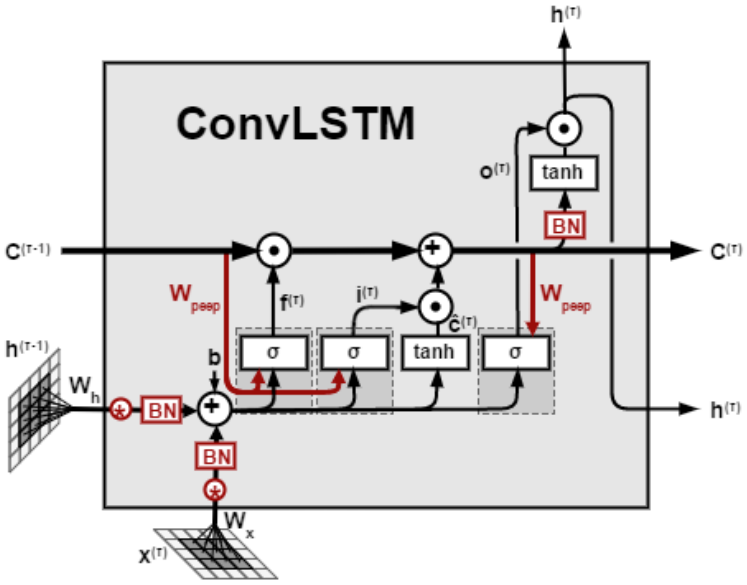

Figure 12: LSTM cell [19]

3.4 Algorithm

3.4.1 Video Processing and Frames Extraction

The main challenge of building a video classifier is finding out a way to feed the videos to a neural network. There are several methods available for this and we followed the traditional approach. Since video is an ordered sequence of frames, we could just extract the frames in order and put them in a 3D sensor so that they can be fed to the neural network. We had different numbers of frames in each video since the number of frames differs from video to video. So we saved them at a fixed interval until the maximum frame was reached.

A sample of frames extracted from a video can be seen in figure 4 below. In actual code, we have stored them in a temporary folder to process and extract features from each frame. These features are stored in a feature vector to feed as input for the neural network.

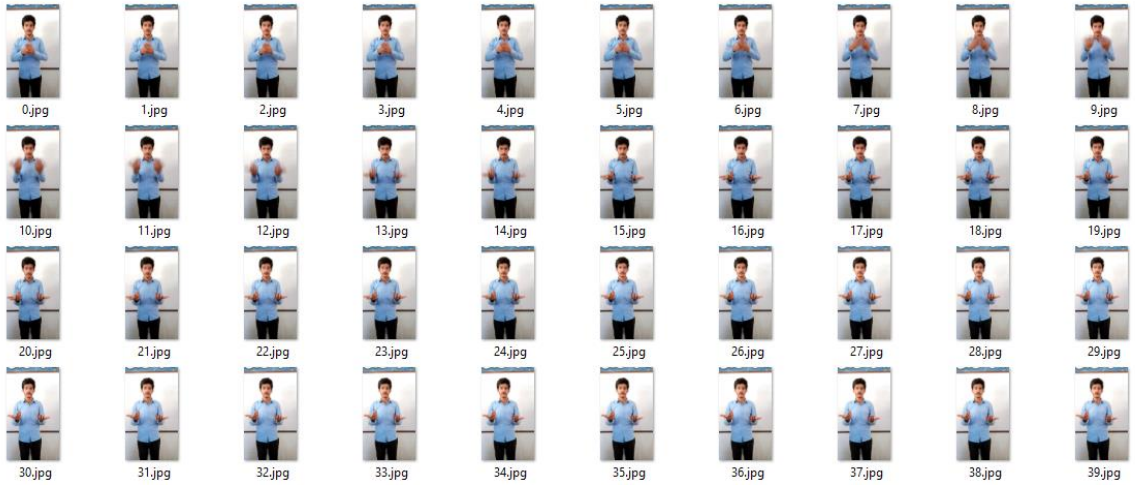

Figure 13: Frames extracted from video sample

We have performed the following steps to process the videos:

1. Capture the frames from the video using the OpenCV library
2. Extract frames from the whole video until the maximum frame is reached.
3. If the video's frame count is less than the maximum frame count, then we need to copy previous frames or use zero padding to match the count length.

3.4.2 Machine Learning Models

For the classification of signs movement and gesture recognition based on captured frames, we needed to train a model on an adequate amount of dataset. For this purpose, we collected our own dataset of sign videos belonging to 29 different categories. We have collected more than 500 sample videos for 29 classes which are not sufficient for the video classification. As we have less amount of sign video datasets so we can even tolerate less accuracy for now. For starting, we extracted features of video frames for a small number of classes. Several machine learning models can be used for classification problems like Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Support Vector Machine (SVM) and Nearest Neighbors (kNN), etc. In this project, we have utilized hybrid architecture which is based on CNN for spatial information and RNN for temporal information. Since we are dealing with time-series data, these are best suited for this problem.

The frames in a video are in an ordered sequence which is related to time series data. We extracted features from the frames and trained our model based on those features. For this purpose, we have utilized the following techniques for feature extraction.

3.4.3 Feature Extraction

3.4.3.1 ConvLSTM

ConvLSTM is a hybrid architecture that is used for a spatial-temporal dataset in which we have a time-series nature of the data based on images. When working with videos, we face challenges of both classification and sequence. The best approach for images is to pass through Convolutional Layers, in which the filters extract important features. In sequential images, we used ConvLSTM layers, that is recurrent neural network as explained earlier, just like the LSTM, but in this model, the internal matrix multiplication is used which are exchanged with convolutional operations. We used this model and passed our frame vector to its layers. The dimensions were kept 3D by the model instead of a 1D array and we got the output as a sequence of all inputs at each time step. We trained this model and checked performance and accuracy on training and testing datasets. We assigned labels on testing videos and calculated probabilities. The label with the highest probability is then predicted along with other classes after that. The problem with this model is that the model was overfitting the dataset. The performance was good on the training dataset but poor on test/unseen data. We opted for different approaches to this model to reduce overfitting. The results are recorded in the next chapter.

3.4.3.2 ResNet50

Keras Application provides many state of the art models, pre-trained on the ImageNet-1k dataset set and these models are very useful in transfer learning on the dataset. These models are mainly used where we have less amount of data for machine learning models. We have used ResNet50, a pre-trained model for our dataset. The meaningful features are extracted from all frames of all training and testing videos and stored in a feature vector. This vector is then passed to the sequential model which is based on CNN-RNN architecture. We have used this hybrid architecture consisting of GRU, and recurrent neural network layers. We trained this model and checked performance and accuracy on training and testing datasets. We assigned labels on testing videos and calculated probabilities. The label with the highest probability is then predicted along with other classes after that. The results are recorded in the next chapter.

3.4.3.3 InceptionV3

The performance of the ResNet50 was not satisfactory as shown in *chapter 4*. We experimented with another model named InceptionV3 which is also a very popular and useful model for extracting features from the image dataset. We passed the frames dataset to this pre-trained InceptionV3 model and extracted features from all frames of all training and testing videos and stored them in a feature vector. This vector is then passed to the sequential model which is based on CNN-RNN architecture. We trained this model and checked performance and accuracy on training and testing datasets. The performance of this model was satisfactory at this level where we do not have an adequate amount of data. We assigned labels on testing videos and calculated probabilities. The label with the highest probability is then predicted along with other classes after that. The results are recorded in the next chapter.

3.5 Server Configuration

Servers are used for different purposes like model deployment, processing and computing power etc. This is the domain of network infrastructure in which we **deploy machine learning models and rely on CPU/GPU/TPU-enabled servers**, flash storage, and other compute infrastructure. This process is done as machine learning models are very processor and storage-intensive. However, the storage systems and servers must be fed data that traverses a network.

There are two types of servers:

1. Web Server (Public Network)
2. Local Server (Private Network)

In our project, we have utilized Local server networking. For this purpose, different Python web frameworks provide local server networking like Flask, Django etc. We have used Flask in this project to establish a backend server connection. A brief description of the Flask framework and related technologies and services is given below:

3.5.1 Flask

Flask is a micro web framework written in Python which depends on the Jinja template engine and the Werkzeug WSGI toolkit [19]. This framework provides useful tools and features that help in creating web and mobile app backend applications in Python. So we have also utilized its services to a local server network architecture.

Figure 14: Flask [19]

3.5.2 Ngrok

Ngrok is an online service that helps to connect the apps with Ngrok's network edge. This is a great platform to put localhost on the internet securely. It can be connected to any system regardless of network or location. We have used this service to put our Flask localhost network online so that we can connect our mobile device with the online network service.

Figure 15: Ngrok [20]

3.6 Android Application

3.6.1 Requirements

To use the application, we have designed and developed the following functionalities which are necessary for the user:

- **Record Video:** The first step is to record the video if the deaf-mute person is in front of you.

- **Select Video:** The user can also select a pre-recorded video from the gallery for sign prediction.

- **Video Visualization/Play**: The second step is video visualization so that the user can see his recorded video and check it before sending it to the server.

- **Server:** The third step is to send a video to the Flask server. The server should be in working condition that will connect the mobile application with the ML model that is deployed on Flask. This server will receive a video file, process it and ML predict the results of this processed video. In the end, it sends the response back to the mobile application.

- **Result Visualization:** Results can be displayed on mobile applications in text form.

- **Audio Sound:** The application should be able to convert the predicted text into speech.

3.6.2 Tools and Service

Multiple tools and services are used in the development of the mobile application for the front-end and back-end which are as follows:

- **Android Studio:** Since we are developing a native application, Android Studio is best suited for the project. It is the official IDE for native android app development. It is used for building the market-leading apps with an intelligent code editor, real-time profilers and emulators and a flexible build system.

- **Flask:** The Flask server is used for the development of infrastructure and connection with the mobile application. The video is sent to the server-side and the response is received and displayed on the mobile interface.

- **Retrofit:** Retrofit is a type-safe HTTP client for Android and Java. Since we needed an HTTP API connection to send video from an android device to the server, this library is used to perform this action. For successful calls and requests, the methods are used and each method must have an HTTP annotation for the request and relative URL. There are eight built-in annotations: *HTTP, GET, POST, PUT, PATCH, DELETE, OPTIONS and HEAD*. [21] We have used the POST method annotation as we have to post the video on the server and receive the response as a result.

3.6.3 Frontend and Backend

The application needs a frontend layout for the user and a backend setup for the processing. This includes:

- **Frontend:**
    1. XML Layout
    2. Java
- **Backend:**
    1. Flask Server
    2. Local Storage
    3. Ngrok API connection

3.6.4 Application Permissions

There are following permissions that the application requires:

1. Internet Connectivity
2. Camera
3. Storage

3.6.5 Application Limitation

The application can be run on android version 4.2 or above.

3.6.6 UI Design

The application has two main screen layouts which are as follows:

- **Start Screen:** The mobile app starts with a one-time splash screen for 3 seconds. It shows the application name ‘SignTalk’ and also displays the app logo as shown in Fig 12.
- **Home Screen:** This is the main screen of the application where all functions are performed. It has a Record button, Select button, VideoView, Predict button and a Sign to Speech button as shown in Fig 13. The user records or selects a video and can play it on video view. After this, the video is sent to the server using the predict button. On successful upload, the app displays the received text response in the TextView section. The user can use the Text to Speech button to play the audio of the received text.

Figure 16: Android App Interface

3.7 Block Diagram

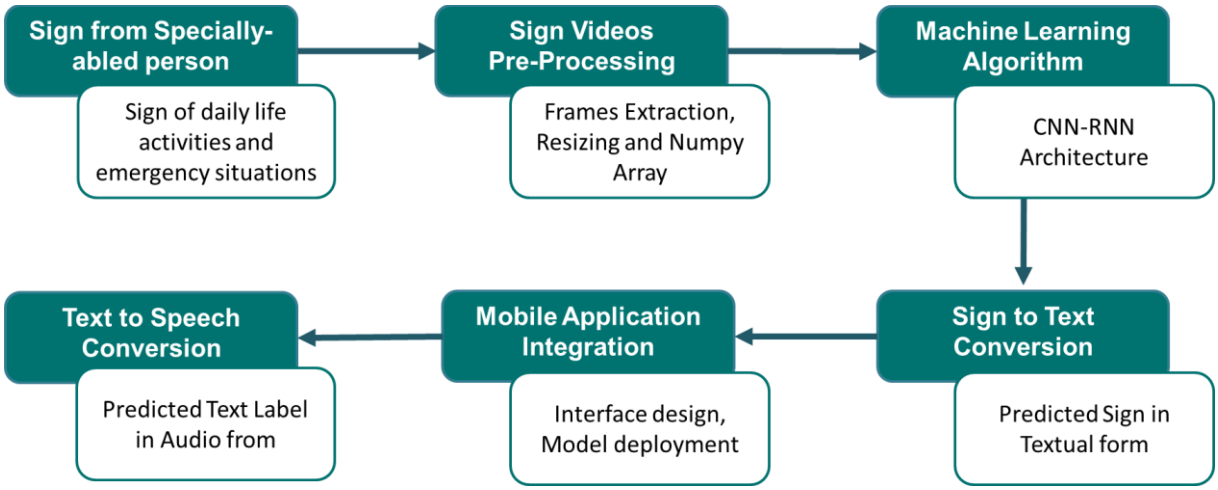

Figure 17: Block Diagram

3.8 Implementation Constraints

1) The main implementation constraint is the dataset in our project. We have collected the dataset manually for machine learning model training but it was not sufficient for the generalization of the project.
2) The mobile application is integrated with the local server network so it is not ready for self-use until we convert it into a product by going through the process of product commercialization.
3) The mobile RAM and local server storage is currently used as a database because of time limitations.
4) Time limitation limits the complete design of the prototype.

Chapter 4

4. Testing

In this project, the CNN-RNN architecture based machine learning models convLSTM, Resnet50 and InceptionV3 have been implemented and evaluated on the training and testing dataset. We have evaluated the performance of each model by computing evaluation metrics i.e. accuracy.

First, we did preprocess of the dataset and trained our models on the sign video dataset. The training and testing accuracies are determined and improved by manipulating hyper parameters according to the dataset. These are listed in *Chapter 5*.

In order to test the system, first, the HTTP connection of the android application and Flask local server is established. After a successful connection, the storage and camera permissions are checked. The user has the option to *Select* or *Record* a video from a mobile camera for prediction using android application buttons. The selected or recorded video is sent to a video view for visualization and then the user can use the *Predict* button to send the video to the Flask server for prediction. The response of the Flask server is sent back to the android application based on prediction results and displayed as text.

The Flask application receives the video file from the android application and performs pre-processing steps which are the same as done before the training of the machine learning model. These steps are also listed in this report *3.3.1* section. This is the necessary process because the pre-processing steps should be the same for the training and testing of the machine learning models. After the processing, the video is evaluated on a machine learning model and predictions results are recorded. The prediction results are assigned labels from the given list of Sentences/Words below:

- “How are you?”
- “I cannot speak”
- “I don’t understand”
- “I am a student”
- “Call the ambulance”
- “Extra class”

The probabilities of each label are calculated with the predicted results. and the label with the highest probability is then selected as response output. This response is then sent to the android application and displayed as text. The user can also use the *Text to Speech* button to convert the predicted text into speech using an android functionality.

Chapter 5

5. Results

5.1 Machine Learning Results

The highest probability of the test video sample is shown in the figures below with the corresponding video displayed. There are also the probabilities of test videos with other classes. Note that the videos are framed that we have just extracted and predicted the model on these frames. We have utilized these frames and converted them into the corresponding video. This is also an indication of validity that our frames are extracted in an ordered sequence.

Now, below are the results of all techniques that we have utilized in our project for sign language prediction.

5.1.1 ConvLSTM

5.1.1.1 Model Accuracy

```
print("Training accuracy: ", history.history['accuracy'][-1]*100)
print("Training Loss: ", history.history['loss'][-1])
print("----------------------------------------")
print("Validation accuracy: ", history.history['val_accuracy'][-1]*100)
print("Validation Loss: ", history.history['val_loss'][-1])
print("----------------------------------------\n")

accuracy = convlstm_model.evaluate(features_test, labels_test, verbose=0);
print(f"Test Accuracy: {round(accuracy[1] * 100, 2)}%")
print(f"Test Loss: {round(accuracy[0], 2)}%")
```

```
Training accuracy:  96.55172228813171
Training Loss:  0.3731014132499695
----------------------------------------
Validation accuracy:  80.7692289352417
Validation Loss:  0.9064874053001404
----------------------------------------

Test Accuracy: 75.0%
Test Loss: 1.02%
```

Figure 18: Accuracy and Loss of Training, Validation and Testing Dataset

The accuracy of the algorithm is also shown in the following table 3:

Table 3: Accuracy and Loss of convLSTM Model

Sr#	Dataset	Accuracy (%)	Loss
1	Training	96.5	0.3
2	Validation	80.7	0.9
3	Testing	75.0	1.0

5.1.1.2 *Plots of Accuracy and Loss*

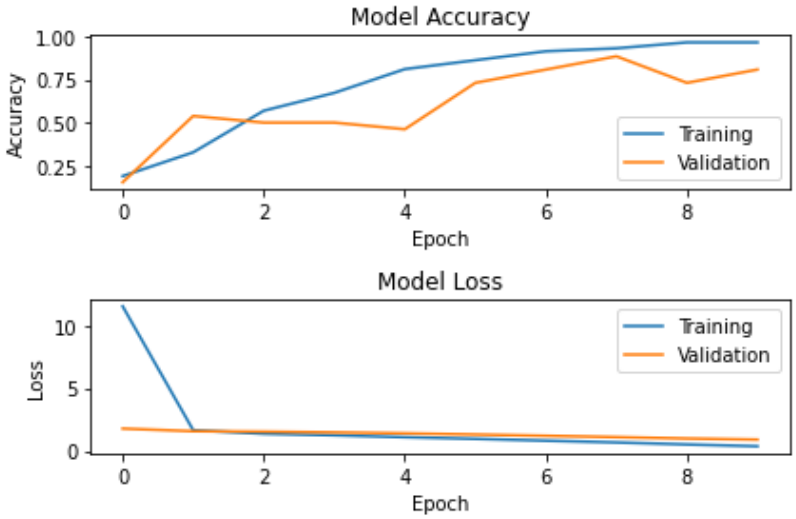

Figure 19: Accuracy and Loss Graph of Training and Validation Dataset

5.1.1.3 *Test Run-1*

```
test_video = '/content/videos_sample_6/i_need_your_help/i_need_your_help-11.mp4'
print(f"Test video path: {test_video}")
test_frames = sequence_prediction(test_video)

to_gif(test_frames[:SEQUENCE_LENGTH])
```

```
Test video path: /content/videos_sample_6/i_need_your_help/i_need_your_help-11.mp4
i_need_your_help
```

Figure 20: Test video predictions for label "i_need_your_help"

5.1.1.4 Test Run-2

```
test_video = '/content/videos_sample_6/call_the_ambulance/call_the_ambulance-2.mp4'
print(f"Test video path: {test_video}")
test_frames = sequence_prediction(test_video)
to_gif(test_frames[:SEQUENCE_LENGTH])
```

```
Test video path: /content/videos_sample_6/call_the_ambulance/call_the_ambulance-2.mp4
call_the_ambulance
```

Figure 21: Test video predictions for label "call_the_ambulance"

5.1.2 ResNet50

5.1.2.1 Model Accuracy

```
print("Training accuracy: ", history.history['accuracy'][-1]*100)
print("Training Loss: ", history.history['loss'][-1])
print("----------------------------------------")
print("Validation accuracy: ", history.history['val_accuracy'][-1]*1
print("Validation Loss: ", history.history['val_loss'][-1])
print("----------------------------------------\n")
_, accuracy = seq_model.evaluate([test_data[0], test_data[1]], test_
print(f"Test accuracy: {round(accuracy * 100, 2)}%")
```

```
Training accuracy:  80.7692289352417
Training Loss:  0.6387647390365601
----------------------------------------
Validation accuracy:  73.9130437374115
Validation Loss:  1.2899788618087769
----------------------------------------

1/1 [==============================] - 0s 226ms/step - loss: 0.9584
Test accuracy: 57.89%
```

Figure 22: Accuracy and Loss of Training, Validation and Testing Dataset

The accuracy of the algorithm is shown in the following table 3:

Table 4: Accuracy and Loss of ResNet50 Model

Sr#	Dataset	Accuracy (%)	Loss
1	Training	80.7	0.6
2	Validation	73.9	1.2
3	Testing	57.8	0.9

5.1.2.2 *Plots of Accuracy and Loss*

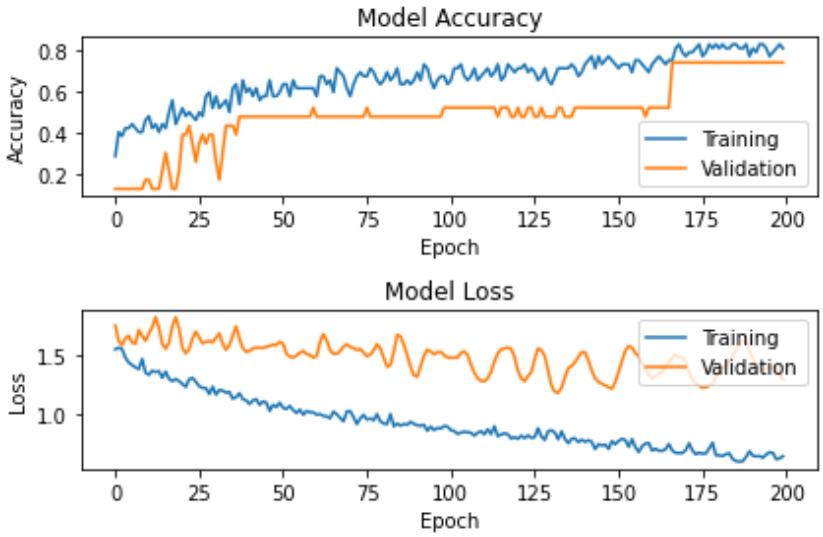

Figure 23: Accuracy and Loss Graph of Training and Validation Dataset

5.1.2.3 *Test Run-1*

```
test_video = '/content/videos_sample_3/i_can_not_speak/i_can_not_speak-16.mp4'
print(f"Test video path: {test_video}")
test_frames = sequence_prediction(test_video)
to_gif(test_frames[:MAX_SEQ_LENGTH])
```

```
Test video path: /content/videos_sample_3/i_can_not_speak/i_can_not_speak-16.mp4
  i_can_not_speak :0: 69.88%
  i_am_a_student :4: 14.86%
  what_is_the_time :3: 10.43%
  thank_you :1:  3.50%
  call_the_ambulance :2:  1.33%
```

Figure 24: Test video predictions for label "i_can_not_speak"

5.1.2.4 Test Run-2

```
test_video = '/content/videos_sample_3/what_is_the_time/what_is_the_time-9.mp4'
print(f"Test video path: {test_video}")
test_frames = sequence_prediction(test_video)
to_gif(test_frames[:MAX_SEQ_LENGTH])
```

```
Test video path: /content/videos_sample_3/what_is_the_time/what_is_the_time-9.mp4
  what_is_the_time :3: 36.46%
  i_can_not_speak :0: 25.11%
  i_am_a_student :4: 16.48%
  thank_you :1: 12.23%
  call_the_ambulance :2:  9.73%
```

Figure 25: Test video predictions for label "what_is_the_time"

5.1.3 InceptionV3

5.1.3.1 Model Accuracy

```
print("Training accuracy: ", history.history['accuracy'][-1]*100)
print("Training Loss: ", history.history['loss'][-1])
print("----------------------------------------")
print("Validation accuracy: ", history.history['val_accuracy'][-1]*100)
print("Validation Loss: ", history.history['val_loss'][-1])
print("----------------------------------------\n")
_, accuracy = seq_model.evaluate([test_data[0], test_data[1]], test_labels);
print(f"Test accuracy: {round(accuracy * 100, 2)}%")
```

```
Training accuracy:  91.72661900520325
Training Loss:  0.3122444748878479
----------------------------------------
Validation accuracy:  82.79569745063782
Validation Loss:  0.8887086510658264
----------------------------------------

3/3 [==============================] - 0s 55ms/step - loss: 0.8007 - accuracy: 0.7742
Test accuracy: 77.42%
```

Figure 26: Accuracy and Loss of Training, Validation and Testing Dataset

The accuracy of the algorithm is shown in the following table 3:

Table 5: Accuracy and Loss of InceptionV3 Model

Sr#	Dataset	Accuracy (%)	Loss
1	Training	91.72	0.31
2	Validation	82.79	0.88
3	Testing	77.42	0.80

5.1.3.2 Plots of Accuracy and Loss

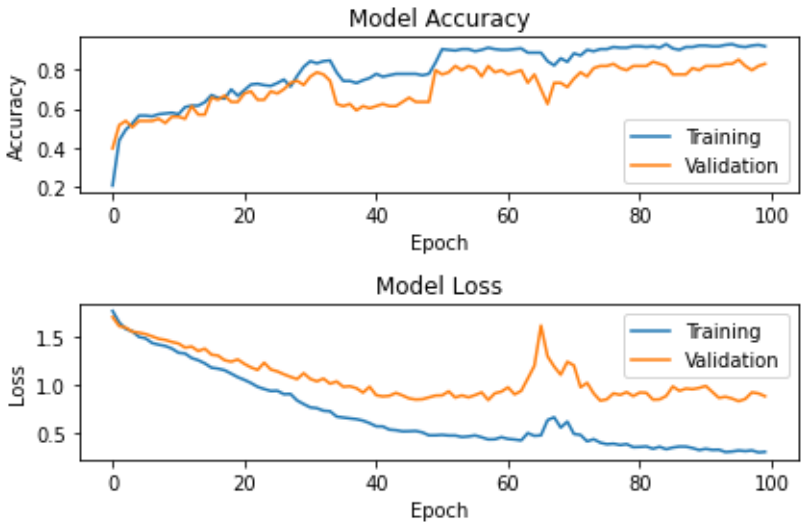

Figure 27: Accuracy and Loss Graph of Training and Validation Dataset

5.1.3.3 Test Run-1

```
Test video path: /content/videos_sample_3/i_am_a_student/
  i_am_a_student :4: 95.60%
  what_is_the_time :3:  1.74%
  i_can_not_speak :0:  1.51%
  thank_you :1:  0.99%
  call_the_ambulance :2:  0.16%
```

Figure 28: Test video predictions for label "i_am_a_student"

5.1.3.4 *Test Run-2*

```
Test video path: /content/videos_sample_3/i_can_not_speak
  i_can_not_speak :0: 89.88%
  call_the_ambulance :2:  5.48%
  i_am_a_student :4:  3.17%
  thank_you :1:  1.10%
  what_is_the_time :3:  0.37%
```

Figure 29: Test video predictions for label "i_can_not_speak"

5.1.3.5 . *Test Run-3 (Wrong Prediction)*

```
test_video = np.random.choice(test_df["path"].values.tolist())
#test_video = '/content/videos_sample/Allah_Hafiz/Allah_Hafiz-11.mp4'
print(f"Test video path: {test_video}")
test_frames = sequence_prediction(test_video)
to_gif(test_frames[:MAX_SEQ_LENGTH])
```

```
Test video path: /content/videos_sample/i_am_a_student/i_am_a_student-5.mp4
  call_the_ambulance :2: 36.35%
  i_am_a_student :4: 36.11%
  Allah_Hafiz :1: 11.31%
  accidental :0:  8.31%
  engineer :3:  7.92%
```

Figure 30: Test video predictions for label "i_am_a_student"

5.2 Mobile Application Results

We have deployed our final model InceptionV3 on Android Application and the results are recorded as shown in figures below:

5.2.1 Test Run for Correct Predictions

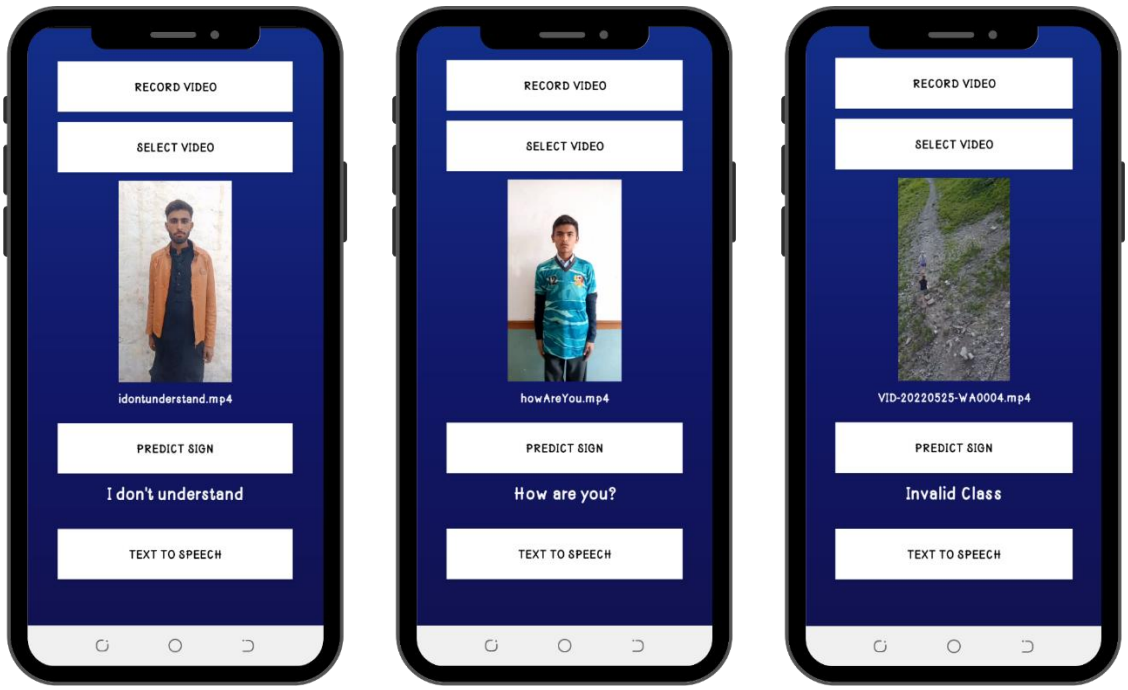

Figure 31: Test video correct predictions on mobile app.

5.2.2 Test Run for Wrong Predictions

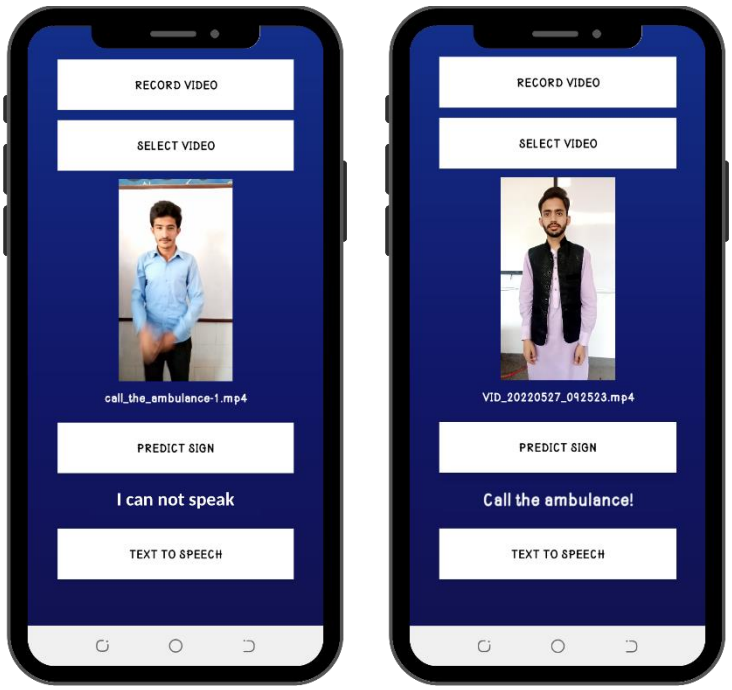

Figure 32: Test video incorrect predictions on mobile app.

Chapter 6

6. Discussion

In our daily life, we communicate with each other using speech and hearing abilities but this is not the case for hearing impaired and inarticulate people. There is a communication barrier that exists between specially-abled and abled people. Because of these communication barriers, signers are facing serious daily life problems including communication and job opportunities etc. But this barrier in society can be minimized if we can design a system that can convert their sign language into speech. For this, we have proposed a solution of designing a mobile app that can convert sign language into speech. As mobile phones are easy to use by anyone at any place by everyone. So this android application would be easy to use for the signers.

So, first of all, we collected our own video dataset for almost 30 classes having signs from daily life words and sentences. The data collection process was time-consuming and therefore we decided to use only common words and sentences. But because of memory limitations and not having advanced GPU systems, we were unable to train models on all classes. So we were able to complete the training of the model on 6 classes only including an extra class for non-sign video prediction.

Before performing the model training, there is a need to preprocess the videos which includes the frames extraction, resizing and normalizing the frames and storing them into the Numpy array so that they can be used for the model training. After that, we developed and worked on machine learning models. For video classification, we have used CNN-RNN architecture. CNN is for feature extraction and spatial processing and RNN to maintain the sequence of frames (temporal processing).

6.1 CNN-RNN Architectures Analysis:

- **Resnet50 Analysis:**
  This pre-trained model was used for the spatial processing and extracting meaningful features from all frames of the training and testing dataset. This feature vector is then passed to the sequential model based on GRU layers. Although the accuracy of the model was 80% on training and 57% on testing the dataset, it failed to generalize the model to the given dataset. As the validation loss was high and the results were not satisfactory on the unseen dataset.

- **InceptionV3 Analysis:**
  After the failure of Resnet50, the next pre-trained model InceptionV3 was used which is a deep learning model based on Convolutional Neural Networks. The same GRU based sequential model was trained on features vectors extracted by the InceptionV3. The accuracy was 91% on the training dataset and 77% on the testing dataset. However, the validation loss and training loss was acceptable as it was below 1.0. The model was performing well even on the unseen dataset.

- **ConvLSTM Analysis:**
  Although the performance was satisfactory on InceptionV3, we also experimented with the ConvLSTM model which was trained on pre-processed video dataset. The whole dataset was divided into training and testing for which the ratio was 70% and 30% for training and testing respectively. The training dataset was given to the model as input and trained on CNN and RNN based architecture for both spatial and temporal information. The training accuracy was recorded at 96.5% and the testing accuracy was 75%. Although the accuracy was higher than the previous results, the testing loss was greater than 1.0, which means that the model was not generalized and performed well on the training dataset only. So it failed to predict unseen data.

The main reason for validation loss and low accuracy was the limited dataset. We have collected our own dataset for the training of the ML model as no data was available online. Since we want to convert the sign language that consists of words and sentences so there is a need to use the video dataset which consists of words and sentences of sign language with hand movements. Video classification is similar to image classification but it is not simple as that. Video processing requires the maintenance of a sequence of frames. The dataset was not sufficient for the generalization of the model and also not for training. But still, we were able to train the model on this limited dataset.

As we can see from model results, some of the results are correct i.e. model is predicting the sign videos as correct while some of the videos are being incorrectly predicated *(Figure 30)*. The reason for this incorrect prediction is because of model performance on unseen data. As we can see testing accuracy is less as compared to model training and validation accuracy i.e. 77%. Because of this the model is predicting incorrect results. The reason for this accuracy is insufficient dataset. If dataset is increased, then there would be better training of model and

hence it would be able to perform well on unseen data as well. While in case of Android app, the similar model is deployed on the local server so for mobile application it also results in incorrect prediction for some of the classes *(Figure 32)*.

After testing the models, we decided to go with the InceptionV3 model as our final choice for sign prediction. The reason for this selection is that the model is more responsive to the unseen dataset than the others. Also, the accuracy of the model can be increased if we have more datasets for the given classes.

The next step was to develop a mobile application for the model and deploy it on the server. We used Flask as backend and Android as frontend technology. The app can be used to record or select a video and send the video to the server for prediction. After successful upload of the video, the Flask server processes the video and evaluates the ML model on this video and finally sends the response back to the android. This response is then displayed on the app screen for the user. The mobile application was limited to the local host only because of time limitations and prototype development. After the successful development of the mobile application, we can use any online server like AWS for processing the model from any public network.

Chapter 7

7. Conclusion

Natural language processing combines linguistics, computer science, and artificial intelligence to examine the interactions between computers and human languages. For specially-abled people, the language of communication is sign language. The phenomenal ability to use machine learning and artificial intelligence has improved the dynamics of the modern world. So we decided to solve the most common problem faced by the specially-abled community in communication by applying the concept of computational theory to real-world problems. In this project, we have introduced an automatic and intelligent system based on mobile application and deep learning that is capable of converting sign language into text and speech with the help of a recorded sign video. The system is based on three main parts i,e; deep learning, mobile application development and server integration. We have experimented with three different deep learning models and results are recorded for each in this report. And finally, the model with the most accurate results and better performance was used for deployment on the Flask server and then integrated with the mobile application. This android application is used to record or select a video and send it to the server for prediction. The server used here is a local network developed with the help of the Flask framework. This server establishes a successful connection of mobile applications with the ML model.

The training and testing of the algorithm is a continuous process and we can increase the dataset for better accuracy of the model so that it has minimum loss when deployed on the mobile application. We have trained the model for a smaller number of classes with more datasets. This means if we increase the dataset per class, we would get much better accuracy of the model and predictions will be done correctly. To this end, we can embed this whole system with a mobile application that can be easily used by the hearing impaired and inarticulate people to convey their message to the ones who don't have any disability. Moreover, people with no hearing and speaking disabilities are also able to use this mobile application to understand the sign language of disabled people.

Chapter 8

8. Future Work

The project is almost at the position from where a little more work will be required to get the best possible accuracy of the machine learning model. The first major task to do is to increase the dataset and record the dataset for more categories and increase the volume of video samples per category. By having this huge dataset, we can train the CNN-RNN model which will increase the accuracy of the system. For this purpose, we will need more processing power and may have to buy online cloud processing services like AWS and Amazon cloud. After getting satisfactory results, we can use the mobile application and work on its architecture to deploy the model successfully on a public network so that people can use this application with ease and portability.

The introduction of facial expressions and whole-body gestures in the identification of signs made by deaf-mute people will significantly improve this Sign Language Recognition System. For a local sign language, a big data collection for a sign language recognition system is a challenging task since it requires learning every sign under every aspect of the moment, hand shape, size, colour, background etc. When we'll be having a good number of datasets, the next step will be to pre-process the video samples so that we can get a cleaned dataset that helps the model to perform even better to give us the correct results. After pre-processing the dataset, the next step would be to implement the pre-trained CNN-RNN model. The model's hyper parameters can be changed to get good accuracy on training, validation, and testing. As a result, we would be able to correctly predict the unseen data from the trained model. Hence by this, we would generalize our model for more classes as well.

After training the CNN-RNN model on more classes a generalized system will be developed. Now the next step would be to embed this whole system with a mobile application to provide the people with hearing and speaking disabilities and the ones with no disabilities, an easy user interface to convey and understand the sign language respectively. The currently developed mobile application can be more efficient in architecture and integration with the server. An easy user interface with a quick response from the server will make the app more efficient and fast. For this purpose, we can use Amazon AWS, Microsoft Azure or other online platforms for model development, processing, deployment and storage requirements. Hence by the end of this phase, we'll be having a commercialized product that can be easily accessible to people.

Chapter 9

9. Reflections on Learning

The final year project helped us in polishing both technical and non-technical skills. This project has helped us to improve the technical skills that we had learned before in some of the courses of doing Electrical Engineering, which includes Introduction to Computing. In this course, we learnt the basics of Python. In our project, we also used the previous concepts with some advanced concepts as well. We used the Python language for machine learning models.

To implement the Machine learning models, we used the concepts of the machine learning course that we studied earlier. While studying this course, we worked over different daily life problems that we can solve through machine learning models by training them over our dataset which includes the image classification and letter classification etc. For this, we used CNN models. So we already had a basic knowledge of image classification but in FYP we have to perform the video classification.

Video classification is almost similar to image classification but not that much easy and simple as image classification is. In video processing, we use the frames extracted from the videos that are basically the images being extracted at specific intervals i.e. frame per second (fps). While processing those frames we have to maintain the sequence of extracted frames. For this, we have used the CNN-RNN model. CNN to extract the features and RNN to maintain the sequence of frames.

While extracting the frames we used the basic concepts of image processing that helped us to perform the pre-processing steps on the video dataset which includes the frame extraction along with resizing and normalization of frames. While implementing machine learning models we also used the basic concepts of image processing while choosing the activation function and applying it. Hence this project was a complete learning experience that helped us to improve our previous knowledge in addition to the new one.

Other than previous subject knowledge enhancement we have also learned new knowledge about the emerging fields the most important one is the NLP. Natural language processing is the most advanced field of research nowadays. It deals with human-computer interaction and natural languages. Along with machine learning, image processing and computer vision we can train a machine in such a way that it acts similarly as the human brain does.

We also came to know about the Android App development that we haven't studied before. Designing an Android app was totally a new experience. First, we got knowledge about the Java programming language so that we can understand what to do while going for the backend. We used the Flask server to upload the model and then connected it with a mobile application. Video is selected from the android app that is processed by the Flask server on which ML model is being deployed and the result of the model is shown on the app screen i.e. in the form of text for the user. Hence this project helped us to learn a new thing that we haven't studied before.

Other than the technical skills, FYP helped to improve the soft skills too. It helped to bring professionalism to our personalities. Because we had dealt with multiple personalities (technical and non-technical) and it helped to improve our communication skills so that we were able to convey our messages clearly and the person in front of us could understand them. The real test of communication skills was during the dataset collection procedure because we have to visit different organizations that are working on sign language. So it was difficult to take time from them by assuring them that we'll complete our task within this time. It brought professionalism to attend the meetings on time and meet the targets that were set in the meeting. Although a few times, targets were not met, supervisors helped to get through that intense situation of work. Other than communication skills, this project has also helped to improve my technical writing skills. Throughout this project, we have studied multiple works of other researchers. And it helped to get an idea of writing a professional technical report, selection of words and the structure of a journal paper.

Hence final year project was completely a new learning process that includes the previous knowledge along with many new concepts and learning that helped us to improve our technical and non-technical skills. So overall it was a good experience.

10. References

[1] “Machine Learning Mastery” https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ (accessed: May. 20, 2022).

[2] “Classification in Machine Learning” www.edureka.co https://www.edureka.co/blog/classification-in-machine-learning (accessed: Apr. 10, 2022)

[3] “Machine Learning regression” www.seldon.io https://www.seldon.io/machine-learning-regression-explained (accessed: May 01, 2022)

[4] “Machine Learning” www.sas.com https://www.sas.com/en_us/insights/analytics/machine-learning.html (accessed: May 02, 2022)

[5] “Clustering Overview” www.developers.google.com https://developers.google.com/machine-learning/clustering/overview (accessed: May 02, 2022)

[6] “What is a Neural Network?” www.aws.amazon.com https://aws.amazon.com/what-is/neural-network/ (accessed: May 09, 2022)

[7] “Introduction to Artificial Neural Network” https://data-flair.training/blogs/wp-content/uploads/sites/2/2019/07/Introduction-to-Artificial-Neural-Networks.jpg (accessed: May 14, 2022)

[8] “Deep Learning Algorithm” www.simplilearn.com https://www.simplilearn.com/tutorials/deep-learning-tutorial/deep-learning-algorithm (accessed: May 16, 2022)

[9] P. D. Rosero-Montalvo et al., "Sign Language Recognition Based on Intelligent Glove using Machine Learning Techniques," in IEEE Third Ecuador Technical Chapters Meeting (ETCM), 2018, pp. 1-5, doi: 10.1109/ETCM.2018.8580268.

[10] Mahesh Kumar, “Conversion of Sign Language into Text” in International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 9 (2018) pp. 7154-7161

[11] Mirzaei, M.R., Ghorshi, S. & Mortazavi, M. “Audio-visual speech recognition techniques in augmented reality environments.” Vis Comput 30, 245–257 (2014). doi: 10.1007/s00371-013-0841-1.

[12] K. Dabre and S. Dholay, "Machine learning model for sign` language interpretation using webcam images," in International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA), 2014, pp. 317-321, doi: 10.1109/CSCITA.2014.6839279.

[13] P. S. M. A. T. D. D. P. Ankit Ojha, " Sign Language to Text and Speech Translation in Real Time Using Convolutional Neural Network," in INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) NCAIT, vol. 8, no. 15, 2020.

[14] Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei, "Large-Scale Video Classification with Convolutional Neural Networks," 2014 IEEE Conference on

Computer Vision and Pattern Recognition, 2014, pp. 1725-1732, doi: 10.1109/CVPR.2014.223.
[15] "Convolutional Neural Network" brilliant.org https://brilliant.org/wiki/convolutional-neural-network/A (accessed: Dec. 21, 2021).
[16] "Basic RNN Structure" researchgate.net https://www.researchgate.net/figure/Basic-RNN-structure_fig1_334464982 (accessed: Jan. 21, 2022).
[17] "GRU" stackoverflow.com https://stackoverflow.com/questions/55261557/understanding-gru-architecture-keras (accessed: Jan. 22, 2022)
[18] "LSTN" www.researchgate.net https://www.researchgate.net/figure/A-typical-architecture-of-a-long-short-term-memory-LSTM-cell-An-LSTM-block-typically_fig1_344213150 (accessed: Jan. 2, 2022)
[19] "Flask Documentation" https://flask.palletsprojects.com/en/2.1.x/ (accessed: May 22, 2022).
[20] "Ngrok" https://ngrok.com/ (accessed: May 22, 2022).
[21] "Retrofit" https://square.github.io/retrofit/ (accessed: May 20, 2022)

11. Appendices

Machine Learning Model - Code:

Library Imports

```
import tensorflow as tf
import tensorflow_hub as hub
import random
import ssl
import cv2
import numpy as np
import imageio
from IPython import display
from urllib import request
import re
import tempfile
import pandas as pd
from keras import backend as K
import sys
import csv
import os
import math
from collections import deque
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras.layers import *
from tensorflow.keras.models import Sequential
from keras.layers import Dense, InputLayer, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, GlobalMaxPooling2D

from keras.layers import LSTM
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D

from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.utils import plot_model
from tensorflow.keras.models import Model
from tensorflow.keras.applications import ResNet50, InceptionResNetV2
from google.colab import drive
import gc
!pip install -q git+https://github.com/tensorflow/docs
```

Mount drive

```
#importing google drive in google colab
drive.mount('/content/gdrive')
```

Define root paths accordingly

```
data_root = '/content/gdrive/MyDrive/sign_videos_sample_a_6'
folder_root = '/content/'
```

CSV creation

```
#creating the Matplotlib figure and specify the size of the figure
plt.figure(figsize=(20,20))

classes = ['call_the_ambulance','i_am_a_student','i_can_not_speak',
'how_are_you', "i_don't_understand", 'extra_class'] # 5 a, 5 b


with open(folder_root+'dataset.csv', 'w', newline='') as file:
  writer = csv.writer(file)
  for c in classes:
    path = os.path.join(data_root, c+"/")
    for i in os.listdir(path):
      writer.writerow([classes.index(c), os.path.join(path, i)])
```

Shuffling CSV data

```
# shuffle the data in csv
df = pd.read_csv(folder_root+'dataset.csv')
ds = df.sample(frac=1)
ds.to_csv(folder_root+'dataset.csv', index=False)
# split the data into train and test

import numpy as np
df = pd.read_csv(folder_root+'dataset.csv', header=None)
df.columns = ["class", "path"]
df = df.astype({"class": str})
train, test = np.split(df.sample(frac=1, random_state=42), [int(.8*len(df))])
```

Defining hyperparameters

```
IMG_SIZE = 224
BATCH_SIZE = 256
EPOCHS = 100
MAX_SEQ_LENGTH = 100
NUM_FEATURES = 2048
```

Count of training and test videos:

```
train_df = train
test_df = test

print(f"Total videos for training: {len(train_df)}")
print(f"Total videos for testing: {len(test_df)}")

train_df.sample(10)
```

Preprocessing Functions

```
def crop_center_square(frame):
    y, x = frame.shape[0:2]
    min_dim = min(y, x)
    start_x = (x // 2) - (min_dim // 2)
    start_y = (y // 2) - (min_dim // 2)
    return frame[start_y : start_y + min_dim, start_x : start_x + min_dim]

def load_video(path, max_frames=0, resize=(IMG_SIZE, IMG_SIZE)):

    cap = cv2.VideoCapture(path)
    cap.set(cv2.CAP_PROP_POS_MSEC, 500)
    frames = []
    j = 0
    try:
        while True:
            ret, frame = cap.read()
            if not ret:
                break
            #frame = crop_center_square(frame)
            frame = cv2.resize(frame, resize)
            frame = frame[:, :, [2, 1, 0]]    #BRG for cv2
            frames.append(frame)
            #cv2.imwrite("/content/frameimg"+str(j)+".jpg", frame)

            if len(frames) == max_frames:
                break
    finally:
        cap.release()
    return np.array(frames)
```

Inception Feature Extractor

```
def build_feature_extractor():
    feature_extractor = keras.applications.InceptionV3(
        weights="imagenet",
```

```
        include_top=False,
        pooling="avg",
        input_shape=(IMG_SIZE, IMG_SIZE, 3),
    )
    preprocess_input = keras.applications.inception_v3.preprocess_input

    inputs = keras.Input((IMG_SIZE, IMG_SIZE, 3))
    preprocessed = preprocess_input(inputs)

    outputs = feature_extractor(preprocessed)
    return keras.Model(inputs, outputs, name="feature_extractor")


feature_extractor = build_feature_extractor()

label_processor = keras.layers.experimental.preprocessing.StringLookup(
    num_oov_indices=0, vocabulary=np.unique(train_df["class"])
)
print(label_processor.get_vocabulary())
class_vocab = label_processor.get_vocabulary()
```

Dataset Preprocessing

```
def prepare_all_videos(df):
    num_samples = len(df)
    video_paths = df["path"].values.tolist()
    labels = df["class"].values
    labels = label_processor(labels[..., None]).numpy()

    # `frame_masks` and `frame_features` are what we will feed to our sequence
model.
    # `frame_masks` will contain a bunch of booleans denoting if a timestep is
    # masked with padding or not.
    frame_masks = np.zeros(shape=(num_samples, MAX_SEQ_LENGTH), dtype="bool")
    frame_features = np.zeros(
        shape=(num_samples, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32"
    )

    # For each video.
    for idx,path in enumerate(video_paths):
        # Gather all its frames and add a batch dimension.hike intern
        #path = video_paths[idx]
        frames = load_video(path)
        frames = frames[None, ...]

        # final_frames = np.append(train_frames, frames) # farmes has 5
dimensions error
```

```
        gc.collect()
        # Initialize placeholders to store the masks and features of the
current video.
        temp_frame_mask = np.zeros(shape=(1, MAX_SEQ_LENGTH,), dtype="bool")
        temp_frame_featutes = np.zeros(
            shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES), dtype="float32"
        )

        # Extract features from the frames of the current video.
        for i, batch in enumerate(frames):
            try:
              video_length = batch.shape[1]
              length = min(MAX_SEQ_LENGTH, video_length)
              for j in range(length):
                temp_frame_featutes[i, j, :] = feature_extractor.predict(
                    batch[None, j, :]
                )
              temp_frame_mask[i, :length] = 1  # 1 = not masked, 0 = masked
              frame_features[idx,] = temp_frame_featutes.squeeze()
              frame_masks[idx,] = temp_frame_mask.squeeze()
            except:
              pass

        gc.collect()
        print(idx)

    return (frame_features, frame_masks), labels

gc.collect()

train_data, train_labels = prepare_all_videos(train_df)

print(f"Frame features in train set: {train_data[0].shape}")
print(f"Frame masks in train set: {train_data[1].shape}")
```

Preparing test data:

```
test_data, test_labels = prepare_all_videos(test_df)

print(f"Frame features in train set: {test_data[0].shape}")
print(f"Frame masks in train set: {test_data[1].shape}")
```

Model building:

```
# Utility for our sequence model.
def get_sequence_model():
```

```
    class_vocab = label_processor.get_vocabulary()

    frame_features_input = keras.Input((MAX_SEQ_LENGTH, NUM_FEATURES))
    mask_input = keras.Input((MAX_SEQ_LENGTH,), dtype="bool")


    x = keras.layers.GRU(16, return_sequences=True, go_backwards=True)(
        frame_features_input, mask=mask_input
    )
    x = keras.layers.GRU(8)(x)
    x = keras.layers.Dropout(0.25)(x)

    x = keras.layers.Dense(8, activation="relu")(x)
    output = keras.layers.Dense(len(class_vocab), activation="softmax")(x)

    rnn_model = keras.Model([frame_features_input, mask_input], output)

    opt = keras.optimizers.Adam(learning_rate=0.001)

    rnn_model.compile(
        loss="sparse_categorical_crossentropy", optimizer=opt,
metrics=["accuracy"]
    )
    return rnn_model
```

Model Training

```
# Utility for running experiments.

filepath = "/tmp/video_classifier"
checkpoint = keras.callbacks.ModelCheckpoint(
    filepath, save_weights_only=True, save_best_only=True, verbose=1
    )
seq_model = get_sequence_model()

history = seq_model.fit(
    [train_data[0], train_data[1]],
    train_labels,
    validation_split=0.25,
    epochs=100,
    callbacks=[checkpoint],
    )
#_, sequence_model = run_experiment()
```

```
seq_model.load_weights(filepath)

print("Training accuracy: ", history.history['accuracy'][-1]*100)
```

```
print("Training Loss: ", history.history['loss'][-1])
print("----------------------------------------")
print("Validation accuracy: ", history.history['val_accuracy'][-1]*100)
print("Validation Loss: ", history.history['val_loss'][-1])
print("----------------------------------------\n")
_, accuracy = seq_model.evaluate([test_data[0], test_data[1]], test_labels);
print(f"Test accuracy: {round(accuracy * 100, 2)}%")

plt.figure(1)

# summarize history for accuracy

plt.subplot(211)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='lower right')

# summarize history for loss

plt.subplot(212)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Training', 'Validation'], loc='upper right')

plt.tight_layout()

plt.show()

sequence_model = seq_model
sequence_model.save(folder_root+'saved_model')
```

 # Load the saved model

```
sequence_model = keras.models.load_model(folder_root+'saved_model')

#sequence_model = keras.models.load_model('inceptionv3.tflite')
```

Inference code for testing on single video

```
import imageio
from tensorflow_docs.vis import embed

def prepare_single_video(path):
       # For each video.
       frames = load_video(path)
       frames = frames[None, ...]

       gc.collect()
       # Initialize placeholders to store the masks and features of the
current video.

       temp_frame_mask = np.zeros(shape=(1, MAX_SEQ_LENGTH,), dtype="bool")
       temp_frame_features = np.zeros(shape=(1, MAX_SEQ_LENGTH, NUM_FEATURES),
dtype="float32")

       # Extract features from the frames of the current video.
       for i, batch in enumerate(frames):
           try:
             video_length = batch.shape[1]
             length = min(MAX_SEQ_LENGTH, video_length)
             for j in range(length):
                temp_frame_features[i, j, :] =
feature_extractor.predict(batch[None, j, :])
             temp_frame_mask[i, :length] = 1  # 1 = not masked, 0 = masked
            #  frame_features = temp_frame_features.squeeze()
            #  frame_masks = temp_frame_mask.squeeze()
           except:
             #print(i, j, length)
             pass

       return temp_frame_features, temp_frame_mask

def sequence_prediction(path):
    class_vocab = label_processor.get_vocabulary()

    frames = load_video(path)
    frame_features, frame_mask = prepare_single_video(path)
    probabilities = sequence_model.predict([frame_features, frame_mask])[0]

    v = ['call_the_ambulance','i_am_a_student','i_can_not_speak',
'how_are_you', "i_don't_understand", 'extra_class'] # 5 a, 5 b

    for i in np.argsort(probabilities)[::-1]: #Add [::-1][:no] for starting
fix number samples
        print(f"  {str(v[class_vocab[i].astype(int)])} :{class_vocab[i]}:
{probabilities[i] * 100:5.2f}%")
    return frames
```

```
def to_gif(images):
    converted_images = images.astype(np.uint8)
    imageio.mimsave("animation.gif", converted_images, fps=25)
    return embed.embed_file("animation.gif")

test_video = np.random.choice(test_df["path"].values.tolist())
#test_video =
'/content/gdrive/MyDrive/sign_videos_sample_a_6/i_can_not_speak/i_can_not_spea
k-16.mp4'
print(f"Test video path: {test_video}")
test_frames = sequence_prediction(test_video)
#to_gif(test_frames[:MAX_SEQ_LENGTH])
```