A

Minor Project Report

on

Sign Language Translator

Submitted for partial fulfillment for the degree of

Bachelor of Technology

(Information Technology)

in

Department of Information Technology

by

Mihika Nigam
189302103

Under the Guidance of
Ms. Kavita

(July-2021)

SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

MANIPAL UNIVERSITY JAIPUR

CERTIFICATE

Date: 17-06-2021

This is to certify that the project titled **Sign language to Audio Translator** is a record of the bonafide work done by **Mihika Nigam** (189302103) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech) in **(Discipline)** of Manipal University Jaipur, during the academic year 2020-21.

Ms. Kavita
Project Guide, Dept of (Information Technology)
Manipal University Jaipur

Dr. Pankaj Vyas
HOD, Dept of (Information Technology)
Manipal University Jaipur

ABSTRACT

Once, sign was viewed as just a system of pictorial gestures with no linguistic structure. In more recent times, researchers have debated that sign is no different from spoken language with all the same linguistic construction. Sign Language Recognition is one among the foremost growing fields of research area. The Sign Language is mainly used for communication by deaf-dumb people. This paper shows the sign language recognizing of 26 hand gestures in American sign language using machine learning algorithms.

By taking an in depth look not only at how sign has been studied over the last 50 years, but also at how the spontaneous gestures that accompany speech are studied. We conclude that signer's gesture just as speakers do. Signers use hand gestures to communicate whereas speakers use words and voice to communicate their thoughts.

In our current growing resources and technical knowledge, we should be helping all the sections of our society. Just like for blind people advanced watches have been developed. Our aim here is to develop a mobile application that lets deaf-dumb people to communicate with their physically abled counterparts. We hope to provide them with a technology that makes them feel more welcomed in the world and not worrying about their disabilities.

We would be creating a machine learning model that recognizes hand gestures. This model would then be used on the flutter application which provides and interface to our model and translates incoming image to text or audio language.

LIST OF TABLES

LIST OF FIGURES

Contents

INTRODUCTION

1.1 *Motivation*

Over 5% of the **world's** population – or 430 million people – require rehabilitation to deal with their 'disabling' hearing loss (432 million adults and 34 million children). It is an estimation that by 2050 one in every ten people will have disabling deafness.

Being deaf can be a tiresome thing in the society. People who are not deaf don't try to learn the sign language to interact with deaf people. Since only a minority of people are deaf, sign language is not taught as a subject in normal schooling. This leads to isolation of deaf people. But if the computer can be programmed in a such a way that it can translate sign language to text format, the difference between the deaf people and the people with normal hearing to be normalized.

Sign language is the main source of communication for the Deaf and Hard-of-Hearing community, but signing is often useful for other groups of individuals also. People with disabilities including Autism, Apraxia of speech, spastic paralysis, and Down Syndrome can also find signing beneficial for communicating.

There is no single sign convention used around the world. Like speech, sign languages developed naturally through different groups of individuals interacting with one another, so there are many sorts. There are somewhere between 138 and 300 different types of signing used around the globe today.

Interestingly, most countries that share an equivalent speech do not necessarily have the same sign language as each other. English for instance, has three varieties: American Sign Language (ASL), British Sign Language (BSL) and Australian Sign Language.

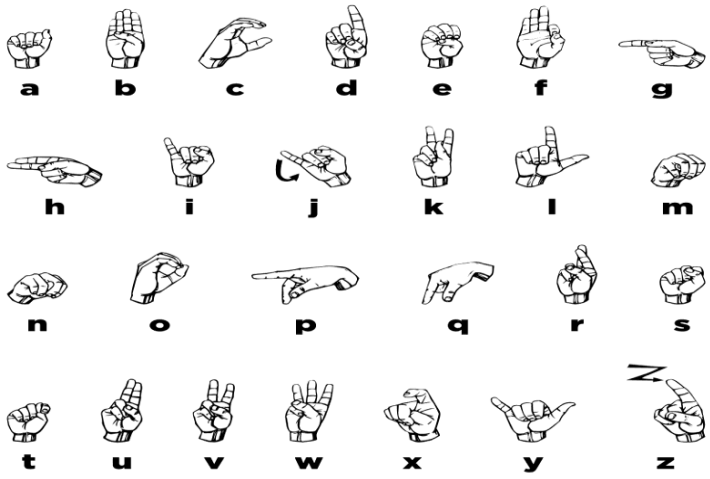

Figure 1. Gestures of sign recognition (ASL)

Most people start their signing journey by learning the A-Z or alphabet equivalent in sign form. The utilizations of the hands to represent individual letters of a written alphabet is named ‘fingerspelling’. It is a crucial tool that helps signers manually spell out names of people, places and things that do not have a long-time sign. We are going to use this similar technology to create our application.

1.2 *Objectives of the project*

The goal of this project was to build a machine learning model able to classify which letter of the American Sign language is being signed, given an image of signing hand. This project also aims to provide a user interface to this ml model.

It is a step towards building a possible signing translator, which may take communications in sign language and translate them into written and oral language. Such a translator would greatly lower the barrier for several deaf and mute individuals to be ready to better communicate with others in day-to-day interactions.

The goal is further motivated by the isolation that is felt within the deaf community. Loneliness and depression exist in higher rates among the deaf population, especially once they are immersed in a world where they can’t communicate in normal ways. Large barriers that profoundly affect life quality stem from the communication disconnect between the deaf and therefore the hearing. Some examples are information deprivation, limitation of social connections, and difficulty integrating in society.

This is in alignment with the motivation as this can make a future implementation of a real time ASL-to-oral/written language translator practical in an everyday situation.

1.3 *Organisation of Report*

This paper is structured as follows.

- Chapter 2 explains the basics and important terminologies required for sentiment analysis and the technologies used.
- Chapter3 explains the method and the approach taken to solve the problem statement.
- Chapter4 contains implementation and results
- Chapter5 contains conclusions and future scope in the same domain.

BACKGROUND OVERVIEW

2.1 *Conceptual Overview*

A various hand gestures were recognized with different methods by different researchers in which were implemented in several fields. The recognition of varied hand gestures was done by vision-based approaches, data glove-based approaches, soft computing approaches like Artificial Neural Network, Fuzzy logic, Genetic Algorithm, et al. like PCA, Canonical Analysis, etc.

Sign language recognition has two different approaches.

- Hardware based approaches

- Machine Learning based approaches.

Hardware based approaches

In this category there are two steps involved: detection of hand gesture and classification into respective alphabet. Several techniques involve hand tracking devices (Leap Motion and Intel Real Sense) and use machine learning algorithms like SVM (Support Vector Machines) to classify the gestures. Hardware devices like kinetic sensors (given by Microsoft) develops a 3D model of the hand and observes the hand movements and their orientations. Another technique was glove based where they require signers to wear a sensor glove or a colored glove. The task is going to be simplified during segmentation process by wearing glove. The disadvantage of this approach is that the signer must wear the sensor hardware alongside the glove during the operation of the system, so it requires tons of setup cost.

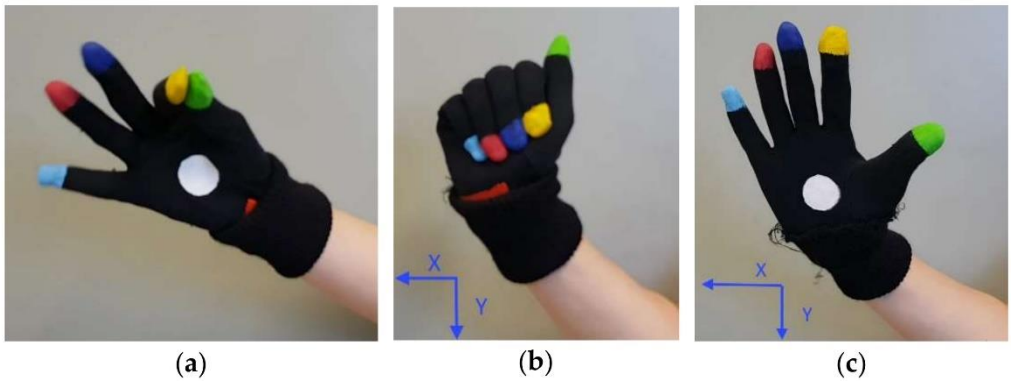

Figure 2. Sensor Gloves

Machine Learning based approaches

Image processing algorithms are utilized in Vision based technique to detect and track hand signs and facial expressions of the signer. This system is easier to the signer since there is no need to

wear any extra hardware. However, there are accuracy issues related to image processing algorithms and these problems are yet to be improved.

There are again two different approaches in vision-based signing recognition:

- **3D model based**: These methods make use of 3D information of key elements of the body parts. Using this information, several important parameters, like palm position, joint angles etc., are often obtained. This approach uses volumetric or skeletal models, or a mixture of the two. Volumetric method is best fitted for computer animation industry and computer vision. This approach is extremely computationally intensive and, systems for live analysis are still to be developed.

- **Appearance based**: Appearance-based systems use images as inputs. They directly interpret from these videos/images. They don't use a spatial representation of the body. The parameters are derived directly from the pictures or videos employing a template database. Some templates are the changeable 2D templates of the human body parts, particularly hands. These sets of points on the outline of an object are known as deformable templates. It's used as an interpolation node for the objects outline approximation.

A few earlier techniques used computer vision techniques just like the convex hull method used to determine the convexities within the image and detect the sting of the hand within the image. There also are contour-based techniques which search for skin-like contours within the image to detect a hand. The detection is done by machine learning algorithms trained for the task of classifying these gestures into alphabets.

The technique used in the project focuses on Neural Network based recognition.

The second part of the project includes creating an application using the tflite model.
TensorFlow Lite is an open-source, cross-platform deep machine learning framework that converts a model pre-trained in TensorFlow to a special format that can be optimized for speed or storage. The special format model are often deployed on edge devices like mobiles using Android or iOS or Linux based embedded devices like Raspberry Pi or Microcontrollers to form the inference at the Edge.

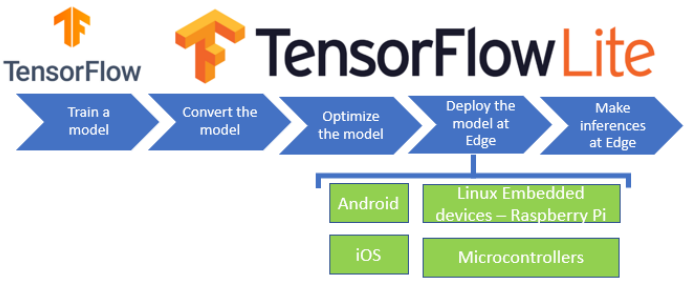

Figure 3. TensorFlow Lite

After the model is trained, we'll convert the Model to the TensorFlow Lite version. TF lite model is a special format model efficient in terms of accuracy and is a light-weight version which will occupy

less space, these features make TF Lite models the correct thing to work on Mobile and Embedded Devices.

2.2 *Technologies Used*

Recommended System Requirements

- Processors: Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAMIntel® Xeon® processor E5-2698 v3 at 2.30 GHz (2 sockets, 16 cores each, 1 thread per core), 64 GB of DRAMIntel® Xeon Phi™ processor 7210 at 1.30 GHz (1 socket, 64 cores, 4 threads per core), 32 GB of DRAM, 16 GB of MCDRAM (flat mode enabled)
- Disk space: 2 to 3 GB
- Operating systems: Windows® 10, macOS*, and Linux*

Minimum System Requirements

- Processors: Intel Atom® processor or Intel® Core™ i3 processor
- Disk space: 1 GB
- Operating systems: Windows* 7 or later, macOS, and Linux
- Python* versions: 2.7.X, 3.6.X

Software Requirement

SPYDER/JUPYTER/PYCHARM or any other software that supports python.
VS CODE with flutter and dart plugins installed.
ANDROID STUDIO with avd manager for application development.

METHODOLOGY

3.1 *Data pre-processing*

The dataset used is a collection of images of hand gestures. The ASL Alphabet data set provides 87000 images of ASL alphabet. Each image is 200x200 pixels. There are 29 classes, of which 26 are the letters A-Z and 3 classes for space, delete and nothing. These three classes are very helpful in real-time applications, and classification. The test data set merely contains 29 images from each class, but to check better accuracy we will be testing it with our own real-world dataset.

Step 1. Since we have our dataset stored in folders, we will be creating a data frame with file paths and labels. Also shuffle the dataset.

Step 2. Training 87000 in a go is difficult without a good gpu. So, we will be training 20000 images at once. Slice the dataset.

Step 3. Create image data generators with necessary parameters. We will be using pre-processing function and validation split parameters.

Step 4. Load the images with a generator.

3.2 *Algorithms Used*

Cnn were designed to connect image data to an output variable. After many researches and practical applications, Cnn have become the best method for any type of prediction problem where image data as an input is involved. The benefit of using Cnn that they can develop an internal representation of a 2-d image. This allows the model to learn position and scale fixed structures in the data.

Our mobile phones have limited memory and computational power, this may affect predictions accuracy and speed of a basic cnn model. To apply the deep convolutional neural network model to real-time applications and low-memory portable devices, a practical solution is to compress and speed up the model to reduce parameters, computation cost, and the power consumption. It is proven that the parameters of deep cnn have a lot of redundancy, and these redundant parameters have little influence on the classification accuracy.

Therefore, in this project we have used MobileNets which is a cnn model based on depthwise separable filters. We will try to understand how a convolutional network works along with the explanation of our MobileNets model.

A typical neural network with no hidden layer is shown in the Figure 4. It simply flattens the input image into a 1D array and uses the input array to help classify the image with activation functions and then gives the potential output where our class of the image could be in.

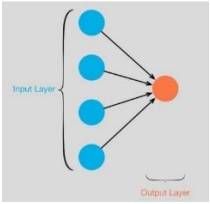

Figure 4. Neural Network without hidden layer

Now this neural network would not have an accuracy of more than 20%. To increase its accuracy, we introduce hidden layers in the model. Convolutions are a set of hidden layers that are used in the neural network architecture. These hidden layers are used to help the computer decide features that could be missed in simply flattening an image into its pixel values. The convolution layers are typically divided into two units: convolutions, and pooling. First, a convolution uses a filter which is applied to the image to highlight certain features deemed important in the classification of the image. These filters can be to bring simple features such as vertical or horizontal lines in spotlight, to make it more obvious to the computer what it is looking at.

These hidden layers help in reducing noise in an image and it highlights the necessary features, and the pooling helps to reduce the size of the image computer has to look at and hence helps in speeding up training time.

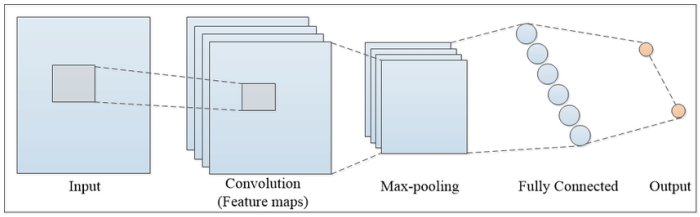

Figure 5. Cnn with hidden convolutional layer

The MobileNet architecture uses depthwise separable convolutions. A typical image is not 2D, in fact it also has depth. This depth is used to represent a color value. For MobileNets the depthwise convolution applies a single filter to each input channel. After this, the pointwise convolution then applies a 1×1 convolution to combine the results of depthwise convolution. In a standard convolution, it filters and combines inputs into a new set of outputs in a single step. The depthwise separable convolution splits this into two layers, an individual layer for filtering and an individual layer for combining. This factorization has the effect of sharply reducing computation and model size.

The architecture of MobileNet can be seen in the Table 1.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

To this model we can also add our additional layers if we want to. From our side we then provide it with the input nodes and give a ‘dense’ function call to finally compile it to the output nodes.

After we have trained and saved our model. We convert our TensorFlow model to TensorFlow Lite model, where the size of our file is reduced.

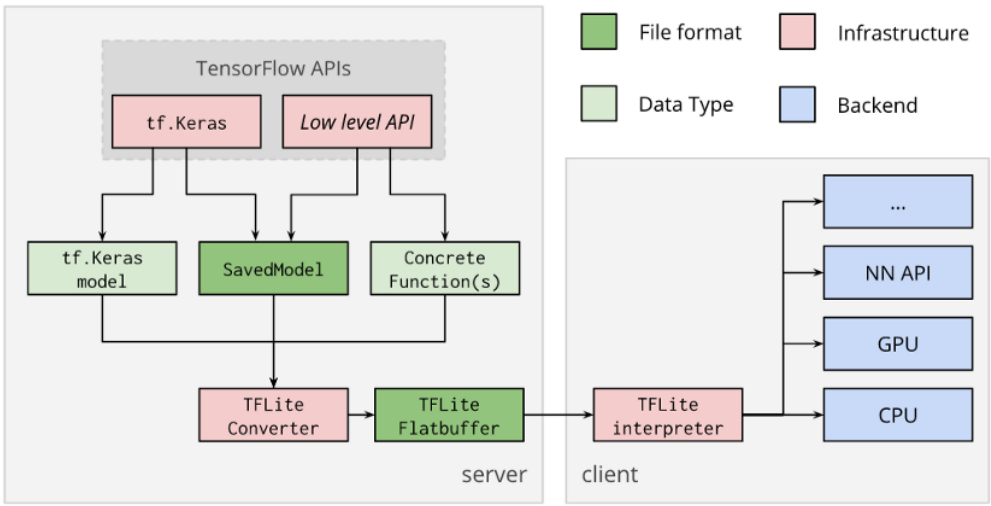

Figure 6. TensorFlow Lite conversion Process

This tflite model is the added to the assets file of flutter application along with the label.txt file.

For the application to understand the model and run it in backend we use the flutter plugin build to help load our model in our application.

Image is then fed to the model, the output given by the model is the displayed by using the UI widget of the application.

IMPLEMENTATION AND RESULTS

4.1 *Algorithm Implementation*

1. Create a data frame with file paths of our dataset. Slice the dataset into testing data and training data.

Out[6]:

	Filepath	Label
0	D:\Downloads\newDB\asl_alphabet_train\asl_alph...	W
1	D:\Downloads\newDB\asl_alphabet_train\asl_alph...	E
2	D:\Downloads\newDB\asl_alphabet_train\asl_alph...	O
3	D:\Downloads\newDB\asl_alphabet_train\asl_alph...	K
4	D:\Downloads\newDB\asl_alphabet_train\asl_alph...	N
...	...	...
86995	D:\Downloads\newDB\asl_alphabet_train\asl_alph...	G
86996	D:\Downloads\newDB\asl_alphabet_train\asl_alph...	nothing
86997	D:\Downloads\newDB\asl_alphabet_train\asl_alph...	N
86998	D:\Downloads\newDB\asl_alphabet_train\asl_alph...	N
86999	D:\Downloads\newDB\asl_alphabet_train\asl_alph...	T

87000 rows × 2 columns

Figure 7. Creating data frame

2. Create generators which has pre-processing parameter where we specify our preprocessing architecture.

In [15]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet.preprocess_input,
    validation_split=0.2
)

test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    preprocessing_function=tf.keras.applications.mobilenet.preprocess_input
)
```

Figure 8. Initializing Data Generators

3. Now we pass our data frame into flow_from_dataframe functions to further derive validation set from training set. We also specify arguments like target_size, color_mode, batch_size etc.,

In [16]:

```
train_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(224,224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=0,
    subset='training'
)

val_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',
    y_col='Label',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=0,
    subset='validation'
)
```

Figure 9. Loading the dataset

4. Now we build our model and fit our dataset to train it.

In [42]:

```
from tensorflow.keras.applications.mobilenet import MobileNet
from tensorflow.keras.layers import Dense, Input, Dropout
from tensorflow.keras.models import Model

baseModel = MobileNet(
    input_shape=(224, 224, 3),
     include_top=False,
     weights='imagenet',
     pooling='avg'
)
baseModel.trainable = False

input_t = baseModel.input

op=Dense(128, activation='relu')(baseModel.output)
op = Dropout(.5)(op) #to prevent overfitting
op =Dense(128, activation='relu')(op)
output_t = Dense(29, activation='softmax')(op)

model = Model(inputs=input_t, outputs=output_t)
```

Figure 10. Building our Model

5. Using predict function, results for test dataset will be predicted.

6. Accuracy is calculated of the predicted images.

7. Save the model and convert it to tflite model

In [98]:

```
model.save('C:/Users/mihika/Desktop/m3.h5')
```

In [8]:

```
from tensorflow.keras import models
model = models.load_model('C:/Users/mihika/Desktop/m3.h5')
```

In [13]:

```
converter =  tf.lite.TFLiteConverter.from_keras_model(model)
tfmodel = converter.convert()
open ("D:/DESKTOP/Minor Project/model.tflite" , "wb") .write(tfmodel)
```

```
INFO:tensorflow:Assets written to: C:\Users\mihika\AppData\Local\Temp\tmpmed_a09b\assets
```

Figure 11. Saving and converting our model

8. Create a flutter project and add all the necessary UI to display output.

9. Load the model.

```
loadModel() async {
  print('im  here');
  await Tflite.loadModel(
    model: "assets/model.tflite",
    labels: "assets/labels.txt",
  );
}
```

Figure 12. Loading model in our mobile application

10. Display the predicted text.

4.2 *Results*

The accuracy came out to be 99% when tested on a dataset which consisted of 4000 images.

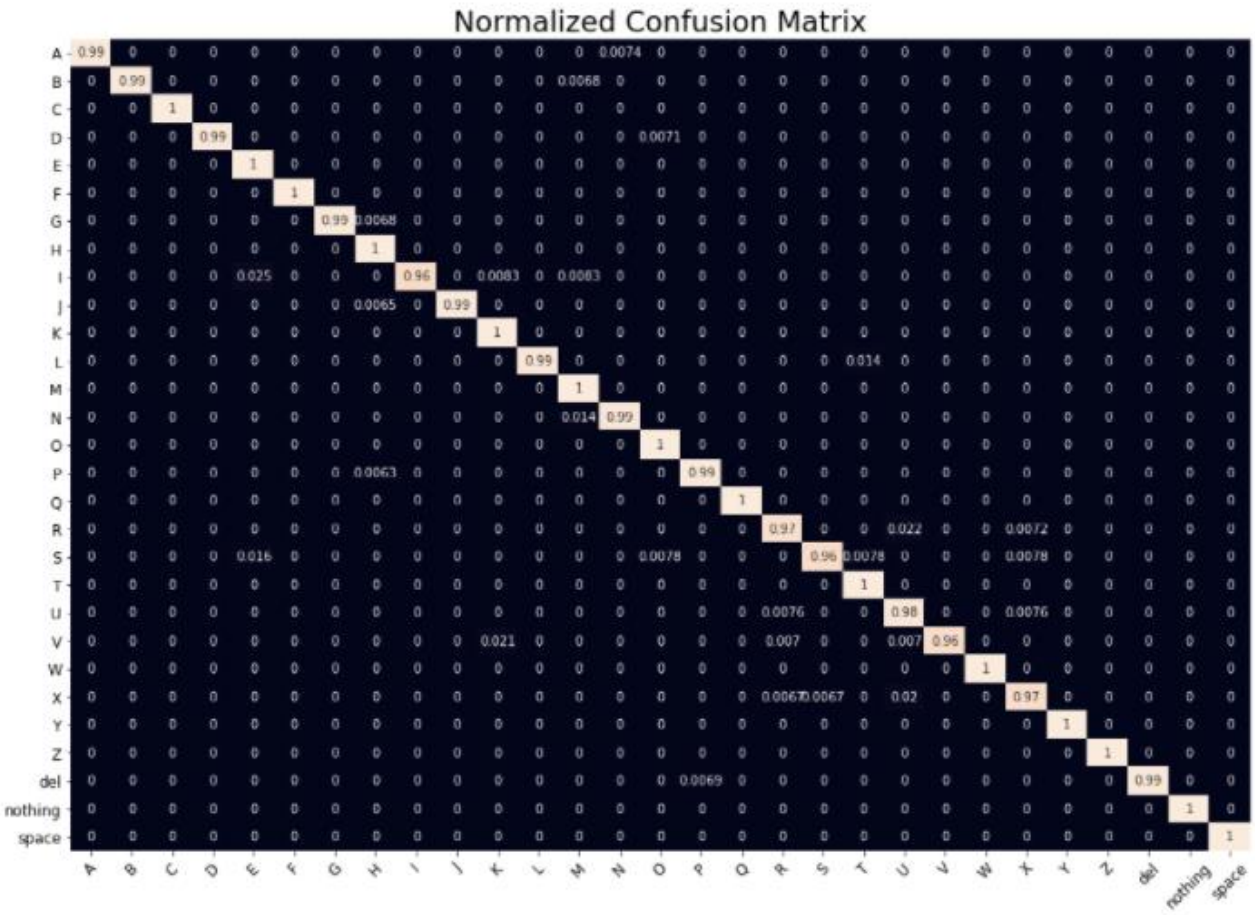

Figure 13. Confusion Matrix

Figure 14. Prediction Plots

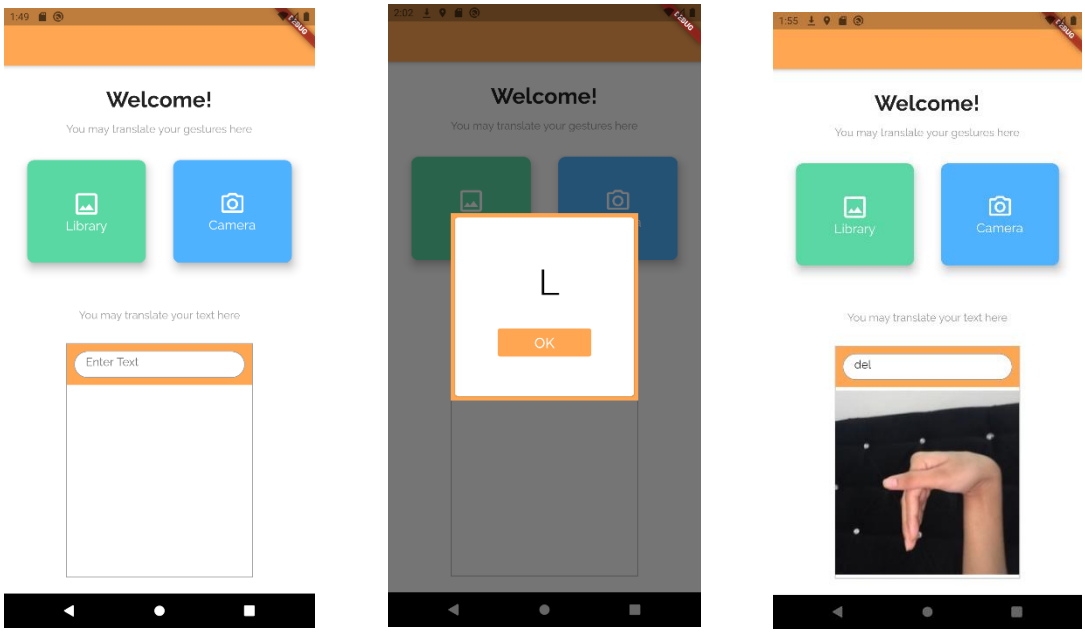

Figure 15. Screenshots from flutter application

FUTURE WORK AND CONCLUSION

5.1 *Timeline Chart*

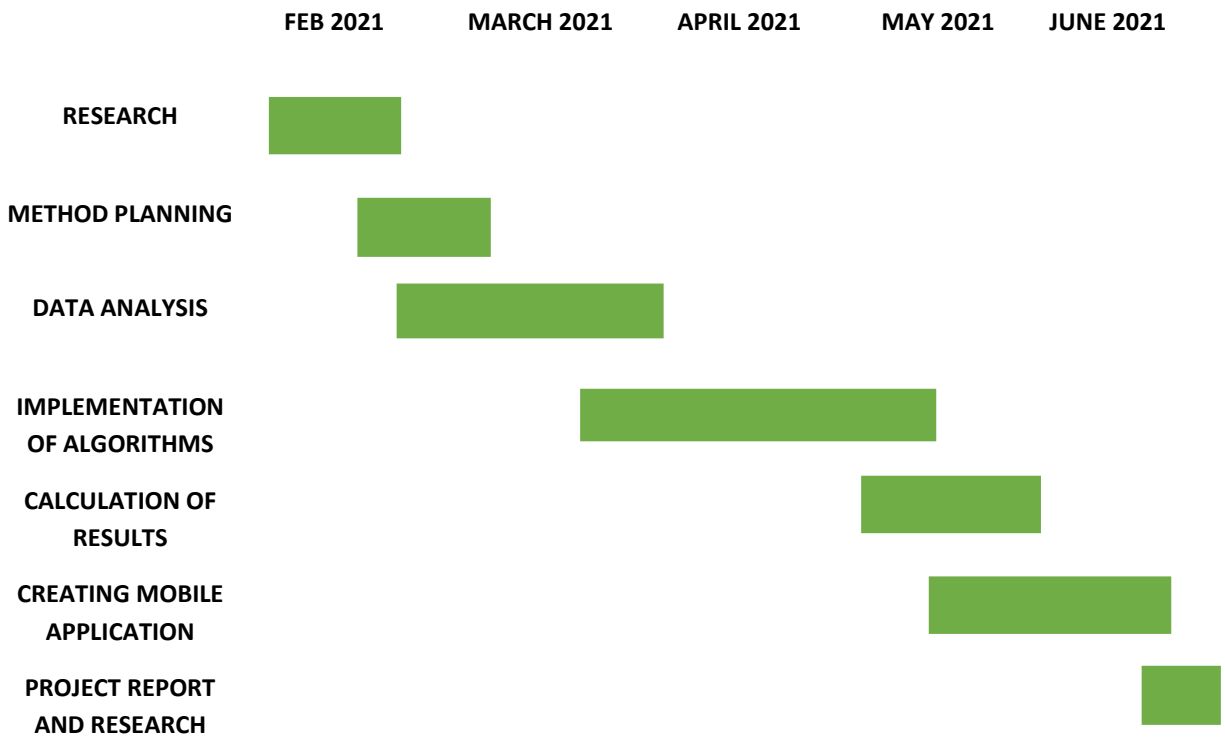

TABLE 2- Timeline Chart

5.2 *Conclusion*

Sign Language recognition system is the bridge to the gap between people with hearing and speech disabilities, and the rest of the society who do not know sign language. This helps in enhancing the harmony in the society and provides better communication for all.

We successfully created an application which translates sign language to normal language. From the results shown in Figure 13, we inferred that our machine learning model MobileNet gave perfect results and it's the best for our application because TF lite models are light-weighted models that can be deployed for a low-latency inference at Edge devices like mobiles, Raspberry Pi, and Micro-controllers. TF lite delegate can be used further. **Delegates** enable hardware acceleration of TensorFlow **Lite** models by leveraging on-device accelerators such as the GPU, to improve the speed, accuracy and power consumption when used.

5.3 *Future Work*

The mobile application can be provided with a functionality where it can detect the input gestures in real time. Or for the more recent updates, a feature can be added where the letters gestured by the user can be accommodated at one place and a word can be formed after the sign is translated. These words can be used to create sentences by using the bag of words concept.

To further improvise the model by training it on dataset which has whole words instead of alphabets.

Since this model only detects ASL, I hope to train my model to detect Indian Sign Language as well, which uses two hands for signing.

A model can also be built on video input and training our model on ID3 algorithm.

REFERENCES (Guideline)

Journal / Conference Papers

[1] Andrew G. Howard and Menglong Zhu, “Mobile Nets”, publication year : 2017

[2] Mahesh Kumar N M, “Conversion of Sign Language into Text”, Volume 13, publication year : 2017, Number 9

Web

[1] Exploring Sign Language Recognition techniques with ML, https://heartbeat.fritz.ai/

[2] www.researchgate.net

[3] A Basic Introduction to TensorFlow Lite, https://towardsdatascience.com/