DAYANANDA SAGAR UNIVERSITY

KUDLU GATE, BANGALORE – 560068

Dept. of CS&E,
School of Engineering
Dayananda Sagar University

Bachelor of Technology
in
COMPUTER SCIENCE AND ENGINEERING

Major Project Phase-II Report

SIGN LANGUAGE TO SPEECH CONVERSION

By
Navaneeth Krishnan Nair – ENG19CS0201
Neha Cochin Narendra – ENG19CS0203
Neha M – ENG19CS0204
Pranav Avinash Mahamuni – ENG19CS0225

Under the supervision of
Prof. Pooja Goud
Asst. Professor, Dept. of CSE

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING,
SCHOOL OF ENGINEERING
DAYANANDA SAGAR UNIVERSITY,
BANGALORE

(2022-2023)

DAYANANDA SAGAR UNIVERSITY

School of Engineering
Department of Computer Science & Engineering
Kudlu Gate, Bangalore – 560068
Karnataka, India

CERTIFICATE

This is to certify that the Phase-II project work titled **"SIGN LANGUAGE TO SPEECH CONVERSION"** is carried out by **Navaneeth Krishnan Nair (ENG19CS0201), Neha Cochin Narendra (ENG19CS0203), Neha M (ENG19CS0204), Pranav Avinash Mahamuni (ENG19CS0225),** bonafide students of Bachelor of Technology in Computer Science and Engineering at the School of Engineering, Dayananda Sagar University, Bangalore in partial fulfillment for the award of degree in Bachelor of Technology in Computer Science and Engineering, during the year **2022-2023**.

Prof. Pooja Goud	**Dr. Girisha G S**	**Dr. Udaya Kumar Reddy K R**
Assistant Professor Dept. of CS&E, School of Engineering Dayananda Sagar University	Chairman CSE School of Engineering Dayananda Sagar University	Dean School of Engineering Dayananda Sagar University
Date:	Date:	Date:

Name of the Examiner **Signature of Examiner**

1.
2.

DECLARATION

We, **Navaneeth Krishnan Nair (ENG19CS0201), Neha Cochin Narendra (ENG19CS0203), Neha M (ENG19CS0204), Pranav Avinash Mahamuni (ENG19CS0225),** are students of eighth semester B. Tech in **Computer Science and Engineering**, at School of Engineering, **Dayananda Sagar University**, hereby declare that the Major Project Stage-II titled **"Sign Language to Speech Conversion"** has been carried out by us and submitted in partial fulfilment for the award of degree in **Bachelor of Technology in Computer Science and Engineering** during the academic year **2022-2023**.

Student	**Signature**
Name1:	
USN :	
Name2	
USN :	
Name3:	
USN :	
Name4:	
USN :	

Place : Bangalore

Date :

ACKNOWLEDGEMENT

It is a great pleasure for us to acknowledge the assistance and support of many individuals who have been responsible for the successful completion of this project work.

First, we take this opportunity to express our sincere gratitude to School of Engineering & Technology, Dayananda Sagar University for providing us with a great opportunity to pursue our Bachelor's degree in this institution.

We would like to thank ***Dr. Udaya Kumar Reddy K R, Dean, School of Engineering & Technology, Dayananda Sagar University*** *for his constant encouragement and expert advice.*

It is a matter of immense pleasure to express our sincere thanks to ***Dr. Girisha G S, Department Chairman***, ***Computer Science and Engineering***, ***Dayananda Sagar University,*** *for providing right academic guidance that made our task possible.*

We would like to thank our guide ***Pooja Goud***, ***Assistant Professor***, ***Dept. of Computer Science and Engineering***, ***Dayananda Sagar University***, *for sparing his/her valuable time to extend help in every step of our project work, which paved the way for smooth progress and fruitful culmination of the project.*

We would like to thank our ***Project Coordinator Dr. Meenakshi Malhotra*** *and* ***Dr. Pramod Kumar Naik*** *as well as all the staff members of Computer Science and Engineering for their support.*

We are also grateful to our family and friends who provided us with every requirement throughout the course.

We would like to thank one and all who directly or indirectly helped us in the Project work.

TABLE OF CONTENTS

ABSTRACT

The hand gestures are one of the typical methods used in sign language. It is very difficult for hearing impaired people to communicate with the world. Hand gestures are a form of non-verbal communication that can be used in several fields such as communication between deaf-mute people, robot control, human-computer interaction, home automation and medical applications. The deaf and hearing impaired make up a sizable community with specific needs that operators and technology have only recently begun to target. There is no such freely available software, let alone a single one with a reasonable price to translate uttered speech into sign language in real time. This project presents a solution that will not only automatically recognize the hand gestures but will also convert it into text and speech output so that impaired people can easily communicate with normal people.

CHAPTER 1

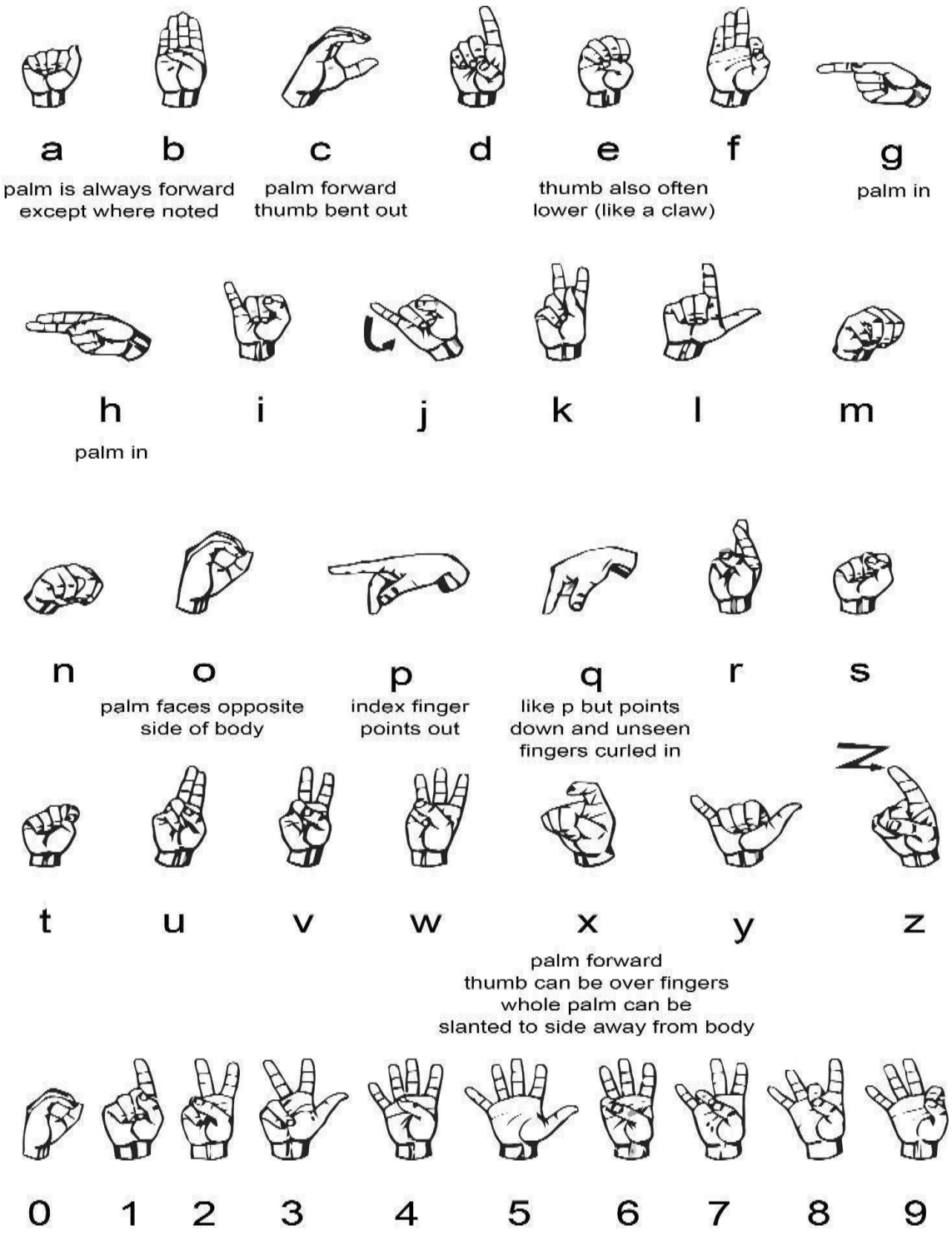
INTRODUCTION

CHAPTER 1

INTRODUCTION

Sign language to speech conversion is a very useful application for people with disabilities to hear and speak. It is an automatic system that enables especially abled people to make a conversation. Communication is a process of exchange of thoughts and information in a number of ways such as visuals, speech , behavior and signals. Deaf and Dumb people use their hands to show different gestures to express their ideas and thoughts to other people. They communicate non verbally and the gestures are understood with vision. Therefore, the nonverbal communication done by the deaf and dumb people is known as sign language. Sign language is a visual language and consists of three components i.e. Fingerspelling, Word level sign vocabulary, Non manual features. As there is no common language for interaction between normal people and deaf and dumb people the sign language conversion acts as a medium for conversation between the two. In our project we shall create a model that will recognize fingerspelling-based gestures and then combine the hand gestures to form the complete word. Since, our project shall give the output in text as well as audio format it is feasible for both dumb and deaf people.

Learning of American Sign Language, Parents are often the source of a child's early acquisition of language, but for children who are deaf, additional people may be models for language acquisition. A deaf child born to parents who are deaf and who already use ASL will begin to acquire ASL as naturally as a hearing child picks up spoken language from hearing parents. However, for a deaf child with hearing parents who have no prior experience with ASL, language may be acquired differently. In fact, 9 out of 10 children who are born deaf are born to parents who hear. Some hearing parents choose to introduce sign language to their deaf children. Hearing parents who choose to have their child learn sign language often learn it along with their child. Children who are deaf and have hearing parents often learn sign language through deaf peers and become fluent.


a
b
c
d
e
f
g
palm is always forward except where noted
palm forward thumb bent out
thumb also often lower (like a claw)
palm in
h
i
j
k
l
m
palm in
n
o
p
q
r
s
palm faces opposite side of body
index finger points out
like p but points down and unseen fingers curled in
t
u
v
w
x
y
z
palm forward thumb can be over fingers whole palm can be slanted to side away from body
0
1
2
3
4
5
6
7
8
9

1.1 FIGURES AND TABLES

1.2 SCOPE

This project will convert the signs only to the English language. We intend to cover the basic English vocabulary used in regular conversation. Inclusion of technical language used in specific fields would complicate and inflate the project. The final product will be a complete application that will read the hand gestures from the user and convey it in an audio format. This application can be used by anyone and will translate sign language to English.

CHAPTER 2
PROBLEM DEFINITION

CHAPTER 2
PROBLEM DEFINITION

Hand gestures can be classified into static and dynamic. As its name implies, the static gesture refers to the stable shape of the hand, whereas the dynamic gesture comprises a series of hand movements such as waving. Hand gestures have a wide range of applications that includes a means of communication for deaf- dumb people. Direct use of hands to show gestures can be a way of natural interaction. Deaf- dumb people need a way of communication with others as sign language is not universally understood. Therefore, our project intends to first convert the sign language into text format and later into audio format for the convenience of the user.

Few Facts about Sign Language that we must know are:

a) Sign Languages have a vital vocabulary as the languages we speak and they also perform all the basic structures that is found in almost all spoken languages.

b) Sign Languages just like spoken languages do not have words that resemble or imitate the sounds.

c) Sign Languages consist of semantics that will guide to describe the meanings of the sign language communications.

d) Unlike other languages methods, sign languages uses the usage of hands to convey meanings.

Quite a lot people rely on sign languages as its their only way to communicate with people, it is also important for the normal crowd to be little mindful of these hand gestures that represent the sign language.

CHAPTER 3

LITERATURE SURVEY

CHAPTER 3
LITERATURE SURVEY

The research paper [1] discusses that hand gestures are one of a non-verbal communication that can be used in fields like robot control, house automation, interactions between people mainly deaf and mute people. This paper performs various hand gestures techniques and tests them and also concludes their advantages, benefits and limitations under various situations. It also calculates its performance of these various methods which mainly focuses on computer vision which includes analyzing, understanding the digital images and extracting the data from the real world. This paper gives a summary of the overview of the gestures with a explanation of its applications.

The research paper [2] discusses that hand gestures can be a great help for people with certain disabilities in interacting with other individuals. This paper proposes a model that is based on YOLO (You Only Look Once) v3and DarkNet53 for the recognition of the gestures. For using YOLOv3 neural networks, first we have to detect the hand gestures in space. By using the above models better accuracy's was achieved such as 97.6, 98.6 and 96.7 percentage. Further, the models were also compared with single shot detector which divides the image using grid and makes each grid detect objects of a particular image, and also VGG16 architecture mainly used for recognition of images. In this paper the trained model can be used for static and dynamic hand images.

The research paper [3] discusses that in today's technology, humans tend to use these hand gestures to communicate with others to convey some meaning or express their feelings. The aim of this paper is to perform a literature review for finding the best suited techniques, applications and the challenges faced in the recognition of hand gestures. It also talks about the feature extraction process, different methods of gestures, hand gesture classifications, recently proposed applications, stating out the challenges while pre-

processing, training and testing the model. They have also included a brief introduction on the recent researches based on hand gesture recognition.

The research paper [4] discusses that in the past, there are many methods used to convert hand gestures into text. But, they had very limited features, performance and capabilities. Few techniques used the glove with the help of sensors which only made the whole process very complex and also very expensive. Few techniques faced difficulties due to the background noise and unwanted disturbances. Few other techniques were dependent on GPU's which made it difficult for a common person to use it. In this paper the data will be extracted and then the analog signals will be converted into digital signals, later the data will be fed into the system and then it identifies the gesture. in this paper the hand gestures will be converted into text and speech. Images will be captured with the help of Open-CV through contour detection techniques.

The research paper [5] discusses that sign Language is one of the oldest forms of languages used for communications. In this paper a real time method using the CNN model is used using the American Sign Language. ASL is a sign language used in the United States of America for the deaf and mute community. It uses both manual parameters such as posture of the hand, motion of the hand, shape of the hand and non-manual parameters that includes like movements of mouth, head, cheeks, shifting the body etc. In this approach the hand gestures are firstly passed through a camera which will be pre-processed again which will be passing through an interpreter which will predict the hand images. An accuracy of 95.8% is achieved and a real time American Sign Language recognition model is been developed for the respective alphabets.

The research paper [6] discusses that Sign Language Recognition is a process that helps the deaf-mute people to communicate with a normal individual. The goal of this paper is to create a model that is very user friendly and accurate a language system of neural

network by producing text and speech as output for the given input gestures. This paper presents a two way interaction as a solution without using any translator. This paper focuses on computer vision based sign gestures using the Convolutional neural network. The dataset used is the American Sign Language. ASL is a sign language used in the United States of America for the deaf and mute community people. The datasets are pre-processed using the python libraries and packages such as OpenCV and skimage which are used for analysing and pre-processing the image patterns, and then it will be trained using the CNN model. The outcome will be converted into speech.

The research paper [7] discusses that hand gestures are one of the oldest methods used in sign language. It is the most commonly used by deaf and dumb people to express, interact among themselves or with others. This paper creates a system that will automatically convert the gestures unto text. In this paper a real time vision based system is used and the main purpose is to recognize the input alphabets that use American Sign Language. In this paper they have achieved an accuracy of 92 % using the American Sign Language dataset. They could also improve their prediction rate by using two layers of algorithms in which they predict the gestures with the help of Gaussian blur filter. It was also pointed out that there was no unwanted disturbances in the background and there was no problem with the lighting too.

The research paper [8] discusses that American Sign Language is considered as one of the oldest sign language which is mainly used for deaf and mute people. This paper utilizes the grey scale and Gaussian blur filter for recognizing and processing an image. In this project, the authors have tried to resolve some of the complex problems which were faced by the deaf-mute impaired people. By using this application an individual can learn and understand the hand gestures and their importance as per the American Sign Language standards. The output of this system would be recognition of hand gesture will be converted into text which will be displayed as output. It will be a great help for hard hearing to interact among themselves and with normal people.

The research paper [9] discusses that Sign Language Recognition is a project that focuses on recognizing the hand gestures and then process them into framing words and sentences. This paper uses "Intel Real Sense" technology which is a depth and tracking technological services, that captures depth images of the hand gestures shown by the user in-front of the camera. The range of the Intel Real Sense camera is up-to 10m. It is mainly used for measuring the objects of any size or shape. The study focuses on developing a assistant digitally that will recognize the hand gestures and then convert them into words or sentences, which will enable people for better communication between the impaired people and common people.

The research paper [10] discusses that a language has various forms and sign language is one such form where communication is done by showing the gestures to express their thoughts, feelings and convey some meanings. This paper makes use of American Sign Language as their dataset. ASL is a natural language that is similar to the spoken language. In this language lots of movements of hands and face will be expressed to show the gesture or symbols. In this method, the image will be captured and will be passed through Gaussian blur filter which help to blur the desired shape or area and removes the unwanted noise and disturbances. After which will be passed through the CNN neural network model that consists of 5 layers to predict the defined class of all the hand symbols. An accuracy of 95% has been achieved for the 26 ASL alphabets thus it helps in assisting for the impaired people.

The research paper [11] discusses that this system recognizes letters and alphabets from Assembly Sign Language Numeric and Character identification. It is a real time American Sign Language application that converts the hand symbols into text as the desired output. Translation of text into speech is done with the help of PYGLET tools which is a powerful and easy to use python library and Google text to speech. In this paper it talks about the

conversion of ASL into human language that can be easily understood so that it will be a great help for the impaired people to feel equal with the normal people. It works using python on JUPYTER and uses vision based method OpenCV. This paper has achieved an accuracy of 99.1% by using the proposed approach testing for 20 frames.

The research paper [12] discusses that Sign language is one of the natural form of language for interacting with people. In this paper a dataset is collected based on segmented RGB image that classifies 36 different symbols or gestures that includes both alphabets and numbers. The system captures the hand gesture as input and the output would be a recognized character in real time. Algorithm used for classification is CNN. Using sign language bridges the gap between the deaf-mute people and normal people. Normal people often face difficulty in communicating with impaired people, hence there is a need for a system that will help them to learn these gestures and use them to interact with them. An accuracy of 98% has been achieved for the proposed approach.

CHAPTER 4
PROJECT DESCRIPTION

CHAPTER 4
PROJECT DESCRIPTION

4.1 IMPLEMENTATION:

1. Whenever the count of a letter detected exceeds a specific value and no other is close to it by a threshold, we print the letter and add it to the current string.
2. Otherwise we clear the current dictionary which has the count of detections of the present symbol to avoid the probability of a wrong letter getting predicted.
3. Whenever the count of a blank detected exceeds a specific value and if the current buffer is empty no spaces are detected.
4. In other cases it predicts the end of a word by printing a space and the current gets appended to the sentence below.

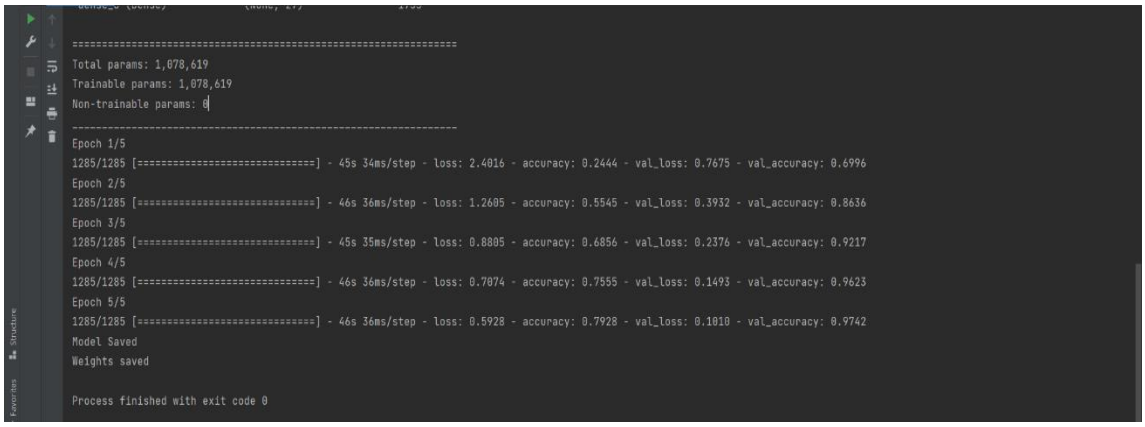


Figure 4.1: Training the model

4.2 System Design:

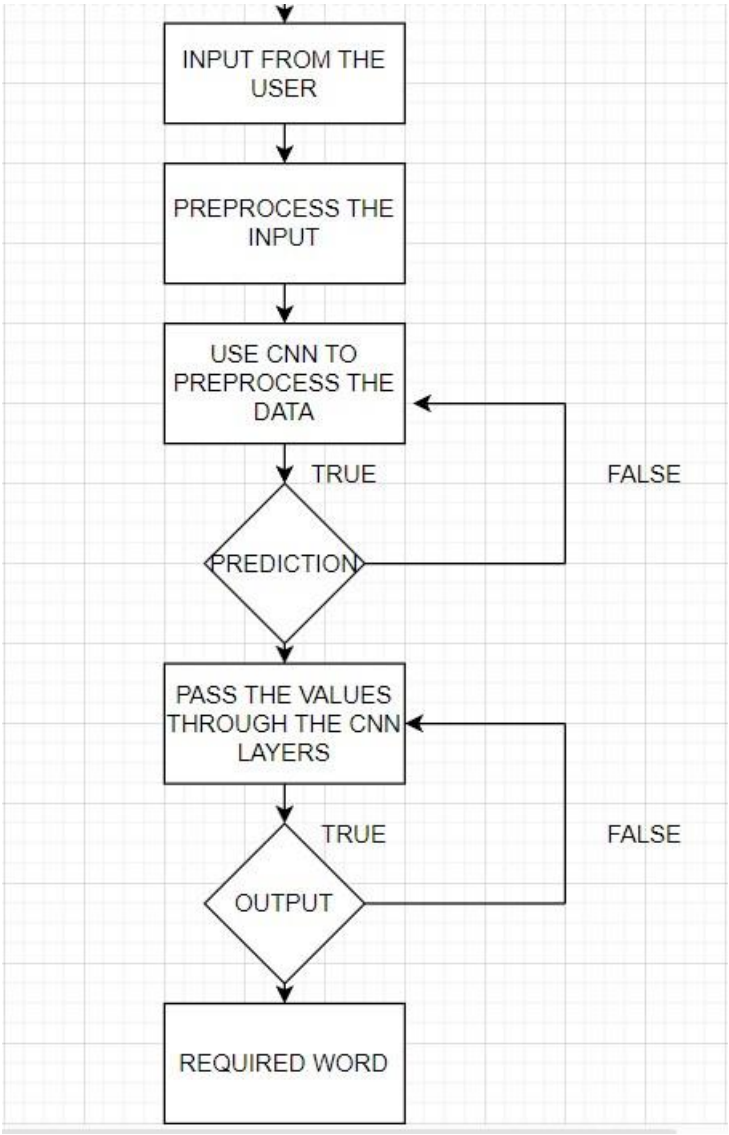


Figure 4.2: System design

4.3 ASSUMPTIONS AND DEPENDENCIES:

- Training data is captured with a camera and pre-processed where gaussian filter is applied to the images to clearly define the boundaries.
- A multi-layer CNN model is used to classify the various images into their respective classes.
- The model is then tested on test images.
- The model will then be linked to an UI that will capture the hand signs in real time and generate the text/speech.

CHAPTER 5

REQUIREMENTS

CHAPTER 5
REQUIREMENTS

5.1 FUNCTIONAL REQUIREMENTS

The Sign Language to Speech convertor should have the following features:

- The system generates the speech from sign language shown by the user.
- The speech generated should be grammatically correct and intelligible
- The final output, that is the speech will be in audio format

5.2 NON-FUNCTIONAL REQUIREMENTS

- Availability - It should be readily accessible and involves simple computation.
- Maintainability - The code can be revised now and then to add new gestures and the accuracy has to be tested periodically.
- Security - As no sensitive data is stored, there are no security and privacy issues.
- Reliability - The application can be trusted to deliver accurate results under any circumstance.

5.3 HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS:

- Web camera
- Processor – Ryzen or Intel
- RAM – 8GB
- Keyboard and Mouse

SOFTWARE REQUIREMENTS:

Python is the programming language used to code. Other tools utilized, to operate this code and to get optimized results are TensorFlow, PyCharm etc.

- PyCharm

  PyCharm is a dedicated Python Integrated Development Environment (IDE) providing a wide range of essential tools for Python developers, tightly integrated to create a convenient environment for productive Python, web, and data science development.

- Keras

  Keras is a high-level neural networks library written in python that works as a wrapper to TensorFlow. It is used in cases where we want to quickly build and test the neural network with minimal lines of code.

- TensorFlow

  TensorFlow is an end-to-end open-source platform for Machine Learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in Machine Learning and developers easily build and deploy Machine Learning powered applications.

- OpenCV

  OpenCV (Open-Source Computer Vision) is an open-source library of programming functions used for real-time computer-vision.

CHAPTER 6
METHODOLOGY

CHAPTER 6
METHODOLOGY

Creation of own dataset

- For the project we tried to find already made datasets but we couldn't find dataset in the form of raw images that matched our requirements.
- All we could find were the datasets in the form of RGB values.
- Hence, we decided to create our own data set.

Gesture classification

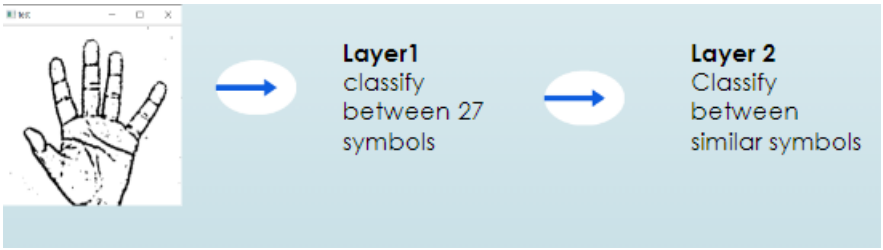


Figure 6.1: Classification

Training and Testing:

We convert our input images (RGB) into grayscale and apply gaussian blur to remove unnecessary noise. We apply adaptive threshold to extract our hand from the background and resize our images to 128 x 128.

We feed the input images after pre-processing to our model for training and testing after applying all the operations mentioned above.

The prediction layer estimates how likely the image will fall under one of the classes. So, the output is normalized between 0 and 1 and such that the sum of each value in each class sums to 1. We have achieved this using SoftMax function.

At first the output of the prediction layer will be somewhat far from the actual value. To make it better we have trained the networks using labelled data. The cross-entropy is a performance measurement used in the classification. It is a continuous function which is positive at values which is not same as labelled value and is zero exactly when it is equal to the labelled value. Therefore, we optimized the cross-entropy by minimizing it as close to zero. To do this in our network layer we adjust the weights of our neural networks. TensorFlow has an inbuilt function to calculate the cross entropy.

As we have found out the cross-entropy function, we have optimized it using Gradient Descent in fact with the best gradient descent optimizer is called Adam Optimizer.

Algorithm Layer 1:

- Apply Gaussian blur filter and threshold to the frame taken with OpenCV to get the processed image after feature extraction.
- This processed image is passed to the CNN model for prediction and if a letter is detected for more than 50 frames then the letter is printed and taken into consideration for forming the word.
- Spaces between the words are considered using the black symbol.

CNN Model:

1. 1st Convolution Layer: The input picture has resolution of 128x128 pixels. It is first processed in the first convolutional layer using 32 filter weights (3x3 pixels each). This will result in a 126X126 pixel image, one for each Filter-weights.
2. 1st Pooling Layer: The pictures are down sampled using max pooling of 2x2 i.e we keep the highest value in the 2x2 square of array. Therefore, our picture is down sampled to 63x63 pixels.
3. 2nd Convolution Layer: Now, these 63 x 63 from the output of the first pooling layer is served as an input to the second convolutional layer. It is processed in the second

convolutional layer using 32 filter weights (3x3 pixels each). This will result in a 60 x 60 pixel image.

4. 2nd Pooling Layer: The resulting images are down sampled again using max pool of 2x2 and is reduced to 30 x 30 resolution of images.
5. 1st Densely Connected Layer: Now these images are used as an input to a fully connected layer with 128 neurons and the output from the second convolutional layer is reshaped to an array of 30x30x32 =28800 values. The input to this layer is an array of 28800 values. The output of these layer is fed to the 2nd Densely Connected Layer. We are using a dropout layer of value 0.5 to avoid overfitting.
6. 2nd Densely Connected Layer: Now the output from the 1st Densely Connected Layer is used as an input to a fully connected layer with 96 neurons.
7. Final layer: The output of the 2nd Densely Connected Layer serves as an input for the final layer which will have the number of neurons as the number of classes we are classifying (alphabets + blank symbol).

Activation Function:

We have used ReLU (Rectified Linear Unit) in each of the layers (convolutional as well as fully connected neurons).

ReLU calculates max(x,0) for each input pixel. This adds nonlinearity to the formula and helps to learn more complicated features. It helps in removing the vanishing gradient problem and speeding up the training by reducing the computation time.

Pooling Layer:

.

We apply **Max** pooling to the input image with a pool size of (2, 2) with ReLU activation function. This reduces the amount of parameters thus lessening the computation cost and reduces over fitting

Dropout Layers:

The problem of overfitting, where after training, the weights of the network are so tuned to the training examples they are given that the network doesn't perform well when given new examples. This layer "drops out" a random set of activations in that layer by setting them to zero. The network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out.

Optimizer:

We have used Adam optimizer for updating the model in response to the output of the loss function.

Adam optimizer combines the advantages of two extensions of two stochastic gradient descent algorithms namely adaptive gradient algorithm (ADA GRAD) and root mean square propagation (RMSProp).

Algorithm Layer 2:

- We detect various sets of symbols which show similar results on getting detected.
- We then classify between those sets using classifiers made for those sets only.
- In our testing we found that following symbols were not showing properly and were giving other symbols also:

1. For D : R and U
2. For U : D and R
3. For I : T,D,K and I
4. For S : M and N

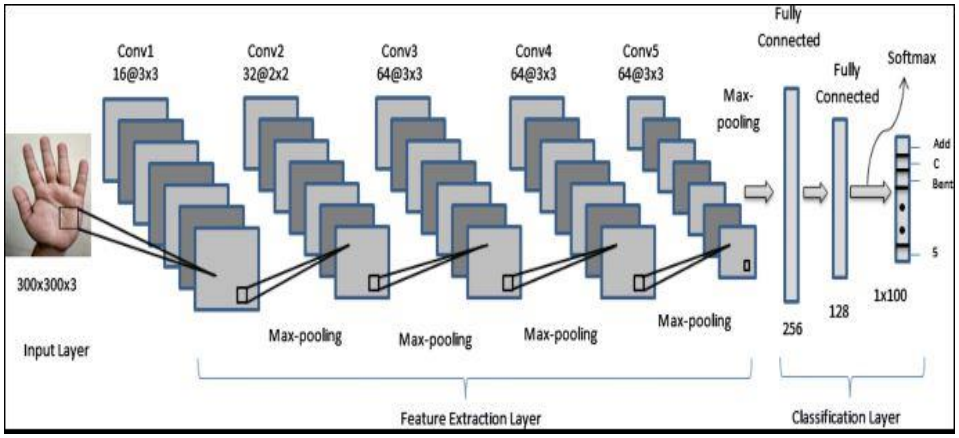


Figure 6.2: CNN model

Finger Spelling Sentence Formation Implementation:

1. Whenever the count of a letter detected exceeds a specific value and no other letter is close to it by a threshold, we print the letter and add it to the current string (In our code we kept the value as 50 and difference threshold as 20).
2. Otherwise, we clear the current dictionary which has the count of detections of present symbol to avoid the probability of a wrong letter getting predicted.
3. Whenever the count of a blank (plain background) detected exceeds a specific value and if the current buffer is empty no spaces are detected.

   In other case it predicts the end of word by printing a space and the current gets appended to the sentence below.

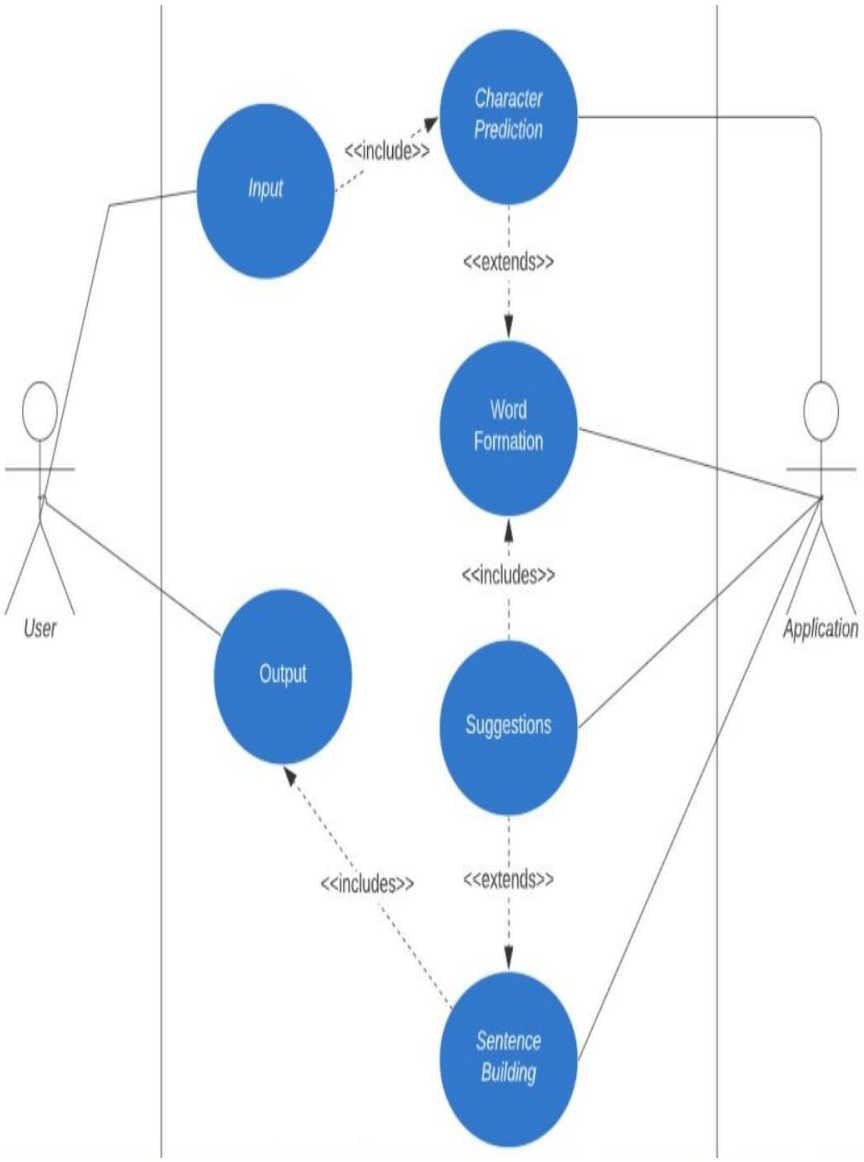


Figure 6.3: Sentence Formation

CHAPTER 7

EXPERIMENTATION

CHAPTER 7
EXPERIMENTATION

Experimental Setup:

The dataset is created for American Sign Language where signs are alphabets of the English language. There are 26 classes, consisting the letters from A-Z. American Sign Language (ASL) is a complete, natural language that has the same linguistic properties as spoken languages, with grammar that differs from English. ASL is expressed by movements of the hands and face.

The experimentation was carried out on a system with a AMD Ryzen 5 5500 U processor, 8 GB memory and webcam and Radeon Graphics , running Windows 11 operating system. The programming environment includes Python (version 3.11), OpenCV (version 4.2.0), TensorFlow Object Detection API.

We have implemented a vision-based approach to our system. Every sign is displayed in front of the camera with an empty bare hand that completely removes the use of electrical or sensor equipment for use. A desktop PC is used and it's placed on the table to detect and track the subjects hand gestures. A clear white background would be preferred for better recognition of the gestures.

- We convert our input images (RGB) into gray scale and apply Gaussian blur to remove unnecessary noise. We apply adaptive threshold to extract our hand from the background and resize our images to 64X64.

- We feed the input images after pre-processing to our model for training and testing after applying all the operations mentioned earlier.

- The prediction layer estimates how likely the image will fall under one of the classes. So, the output is normalized between 0 and 1 and such that the sum of each value in each class sums to 1. We have achieved this using SoftMax function.

While implementing our model we have encountered the problem of accuracy and we are trying to improve the accuracy by increasing the epoch and increasing the data set size.

Collecting data:

In [ ]:

```
import cv2
import numpy as np
import os
import string
# Create the directory structure
if not os.path.exists("data"):
    os.makedirs("data")
if not os.path.exists("data/train"):
    os.makedirs("data/train")
if not os.path.exists("data/test"):
    os.makedirs("data/test")
for i in range(3):
    if not os.path.exists("data/train/" + str(i)):
        os.makedirs("data/train/"+str(i))
    if not os.path.exists("data/test/" + str(i)):
        os.makedirs("data/test/"+str(i))

for i in string.ascii_uppercase:
    if not os.path.exists("data/train/" + i):
        os.makedirs("data/train/"+i)
    if not os.path.exists("data/test/" + i):
        os.makedirs("data/test/"+i)
# Train or test
mode = 'train'
directory = 'data/'+mode+'/'
minValue = 70
cap = cv2.VideoCapture(0)
interrupt = -1
while True:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)
    # Getting count of existing images
```

```
count = {
         'zero': len(os.listdir(directory+"/0")),
         'one': len(os.listdir(directory+"/1")),
         'two': len(os.listdir(directory+"/2")),
         'a': len(os.listdir(directory+"/A")),
         'b': len(os.listdir(directory+"/B")),
         'c': len(os.listdir(directory+"/C")),
         'd': len(os.listdir(directory+"/D")),
         'e': len(os.listdir(directory+"/E")),
         'f': len(os.listdir(directory+"/F")),
         'g': len(os.listdir(directory+"/G")),
         'h': len(os.listdir(directory+"/H")),
         'i': len(os.listdir(directory+"/I")),
         'j': len(os.listdir(directory+"/J")),
         'k': len(os.listdir(directory+"/K")),
         'l': len(os.listdir(directory+"/L")),
         'm': len(os.listdir(directory+"/M")),
         'n': len(os.listdir(directory+"/N")),
         'o': len(os.listdir(directory+"/O")),
         'p': len(os.listdir(directory+"/P")),
         'q': len(os.listdir(directory+"/Q")),
         'r': len(os.listdir(directory+"/R")),
         's': len(os.listdir(directory+"/S")),
         't': len(os.listdir(directory+"/T")),
         'u': len(os.listdir(directory+"/U")),
         'v': len(os.listdir(directory+"/V")),
         'w': len(os.listdir(directory+"/W")),
         'x': len(os.listdir(directory+"/X")),
         'y': len(os.listdir(directory+"/Y")),
         'z': len(os.listdir(directory+"/Z"))
         }

# Printing the count in each set to the screen
# cv2.putText(frame, "MODE : "+mode, (10, 50), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
# cv2.putText(frame, "IMAGE COUNT", (10, ), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
```

```
cv2.putText(frame, "ZERO : "+str(count['zero']), (10, 70), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "ONE : "+str(count['one']), (10, 80), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "TWO : "+str(count['two']), (10, 90), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
# cv2.putText(frame, "THREE : "+str(count['three']), (10, 180), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
# cv2.putText(frame, "FOUR : "+str(count['four']), (10, 200), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
# cv2.putText(frame, "FIVE : "+str(count['five']), (10, 220), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
# cv2.putText(frame, "SIX : "+str(count['six']), (10, 230), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "a : "+str(count['a']), (10, 100), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "b : "+str(count['b']), (10, 110), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "c : "+str(count['c']), (10, 120), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "d : "+str(count['d']), (10, 130), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "e : "+str(count['e']), (10, 140), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "f : "+str(count['f']), (10, 150), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "g : "+str(count['g']), (10, 160), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "h : "+str(count['h']), (10, 170), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "i : "+str(count['i']), (10, 180), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "j : "+str(count['j']), (10, 190), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "k : "+str(count['k']), (10, 200), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "l : "+str(count['l']), (10, 210), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "m : "+str(count['m']), (10, 220), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "n : "+str(count['n']), (10, 230), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "o : "+str(count['o']), (10, 240), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "p : "+str(count['p']), (10, 250), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "q : "+str(count['q']), (10, 260), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "r : "+str(count['r']), (10, 270), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "s : "+str(count['s']), (10, 280), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "t : "+str(count['t']), (10, 290), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "u : "+str(count['u']), (10, 300), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "v : "+str(count['v']), (10, 310), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "w : "+str(count['w']), (10, 320), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "x : "+str(count['x']), (10, 330), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "y : "+str(count['y']), (10, 340), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
cv2.putText(frame, "z : "+str(count['z']), (10, 350), cv2.FONT_HERSHEY_PLAIN, 1, (0,255,255), 1)
# Coordinates of the ROI
x1 = int(0.5*frame.shape[1])
```

```
    y1 = 10
    x2 = frame.shape[1]-10
    y2 = int(0.5*frame.shape[1])
    # Drawing the ROI
    # The increment/decrement by 1 is to compensate for the bounding box
    cv2.rectangle(frame, (220-1, 9), (620+1, 419), (255,0,0) ,1)
    # Extracting the ROI
    roi = frame[10:410, 220:520]
#    roi = cv2.resize(roi, (64, 64))

    cv2.imshow("Frame", frame)
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

    blur = cv2.GaussianBlur(gray,(5,5),2)
    # #blur = cv2.bilateralFilter(roi,9,75,75)

    th3 = cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,11,2)
    ret, test_image = cv2.threshold(th3, minValue, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    #time.sleep(5)
    #cv2.imwrite("/home/rc/Downloads/soe/im1.jpg", roi)
    #test_image = func("/home/rc/Downloads/soe/im1.jpg")



    test_image = cv2.resize(test_image, (300,300))
    cv2.imshow("test", test_image)

    #_, mask = cv2.threshold(mask, 200, 255, cv2.THRESH_BINARY)
    #kernel = np.ones((1, 1), np.uint8)
    #img = cv2.dilate(mask, kernel, iterations=1)
    #img = cv2.erode(mask, kernel, iterations=1)
    # do the processing after capturing the image!
#    roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
#    _, roi = cv2.threshold(roi, 120, 255, cv2.THRESH_BINARY)
```

```
##gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
##cv2.imshow("GrayScale", gray)
##blur = cv2.GaussianBlur(gray,(5,5),2)

#blur = cv2.bilateralFilter(roi,9,75,75)

##th3 = cv2.adaptiveThreshold(frame,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,11,2)
##ret, res = cv2.threshold(th3, minValue, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
# cv2.imshow("ROI", roi)
#roi = frame
interrupt = cv2.waitKey(10)
if interrupt & 0xFF == 27: # esc key
    break
if interrupt & 0xFF == ord('0'):
    cv2.imwrite(directory+'0/'+str(count['zero'])+'.jpg', roi)
if interrupt & 0xFF == ord('1'):
    cv2.imwrite(directory+'1/'+str(count['one'])+'.jpg', roi)
if interrupt & 0xFF == ord('2'):
    cv2.imwrite(directory+'2/'+str(count['two'])+'.jpg', roi)
# if interrupt & 0xFF == ord('3'):
#     cv2.imwrite(directory+'3/'+str(count['three'])+'.jpg', roi)
# if interrupt & 0xFF == ord('4'):
#     cv2.imwrite(directory+'4/'+str(count['four'])+'.jpg', roi)
# if interrupt & 0xFF == ord('5'):
#     cv2.imwrite(directory+'5/'+str(count['five'])+'.jpg', roi)
# if interrupt & 0xFF == ord('6'):
#     cv2.imwrite(directory+'6/'+str(count['six'])+'.jpg', roi)
if interrupt & 0xFF == ord('a'):
    cv2.imwrite(directory+'A/'+str(count['a'])+'.jpg', roi)
if interrupt & 0xFF == ord('b'):
    cv2.imwrite(directory+'B/'+str(count['b'])+'.jpg', roi)
if interrupt & 0xFF == ord('c'):
    cv2.imwrite(directory+'C/'+str(count['c'])+'.jpg', roi)
if interrupt & 0xFF == ord('d'):
    cv2.imwrite(directory+'D/'+str(count['d'])+'.jpg', roi)
```

```
        cv2.imwrite(directory+'E/'+str(count['e'])+'.jpg', roi)
    if interrupt & 0xFF == ord('f'):
        cv2.imwrite(directory+'F/'+str(count['f'])+'.jpg', roi)
    if interrupt & 0xFF == ord('g'):
        cv2.imwrite(directory+'G/'+str(count['g'])+'.jpg', roi)
    if interrupt & 0xFF == ord('h'):
        cv2.imwrite(directory+'H/'+str(count['h'])+'.jpg', roi)
    if interrupt & 0xFF == ord('i'):
        cv2.imwrite(directory+'I/'+str(count['i'])+'.jpg', roi)
    if interrupt & 0xFF == ord('j'):
        cv2.imwrite(directory+'J/'+str(count['j'])+'.jpg', roi)
    if interrupt & 0xFF == ord('k'):
        cv2.imwrite(directory+'K/'+str(count['k'])+'.jpg', roi)
    if interrupt & 0xFF == ord('l'):
        cv2.imwrite(directory+'L/'+str(count['l'])+'.jpg', roi)
    if interrupt & 0xFF == ord('m'):
        cv2.imwrite(directory+'M/'+str(count['m'])+'.jpg', roi)
    if interrupt & 0xFF == ord('n'):
        cv2.imwrite(directory+'N/'+str(count['n'])+'.jpg', roi)
    if interrupt & 0xFF == ord('o'):
        cv2.imwrite(directory+'O/'+str(count['o'])+'.jpg', roi)
    if interrupt & 0xFF == ord('p'):
        cv2.imwrite(directory+'P/'+str(count['p'])+'.jpg', roi)
    if interrupt & 0xFF == ord('q'):
        cv2.imwrite(directory+'Q/'+str(count['q'])+'.jpg', roi)
    if interrupt & 0xFF == ord('r'):
        cv2.imwrite(directory+'R/'+str(count['r'])+'.jpg', roi)
    if interrupt & 0xFF == ord('s'):
        cv2.imwrite(directory+'S/'+str(count['s'])+'.jpg', roi)
    if interrupt & 0xFF == ord('t'):
        cv2.imwrite(directory+'T/'+str(count['t'])+'.jpg', roi)
    if interrupt & 0xFF == ord('u'):
        cv2.imwrite(directory+'U/'+str(count['u'])+'.jpg', roi)
    if interrupt & 0xFF == ord('v'):
        cv2.imwrite(directory+'V/'+str(count['v'])+'.jpg', roi)
```

Image Preprocessing:

```
import numpy as np
import cv2
minValue = 70
def func(path):
    frame = cv2.imread(path)

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray,(5,5),2)

    th3 = cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_INV,11,2)
    ret, res = cv2.threshold(th3, minValue, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
    return res
```

Preprocessing:

```
import numpy as np
import cv2
import os
import csv
from image_processing import func
if not os.path.exists("data2"):
    os.makedirs("data2")
if not os.path.exists("data2/train"):
    os.makedirs("dataluv/train")
if not os.path.exists("data2/test"):
    os.makedirs("data2/test")
path="train"
path1 = "data2"
a=['label']
for i in range(64*64):
    a.append("pixel"+str(i))
#outputLine = a.tolist()
label=0
var = 0
c1 = 0
c2 = 0
for (dirpath,dirnames,filenames) in os.walk(path):
    for dirname in dirnames:
        print(dirname)
        for(direcpath,direcnames,files) in os.walk(path+"/"+dirname):
               if not os.path.exists(path1+"/train/"+dirname):
                os.makedirs(path1+"/train/"+dirname)
            if not os.path.exists(path1+"/test/"+dirname):
                os.makedirs(path1+"/test/"+dirname)
            # num=0.75*len(files)
            num = 100000000000000000
            i=0
            for file in files:
                var+=1
                actual_path=path+"/"+dirname+"/"+file
```

```
                actual_path2=path1+"/"+"test/"+dirname+"/"+file
                img = cv2.imread(actual_path, 0)
                bw_image = func(actual_path)
                if i<num:
                    c1 += 1
                    cv2.imwrite(actual_path1 , bw_image)
                else:
                    c2 += 1
                    cv2.imwrite(actual_path2 , bw_image)
                i=i+1
        label=label+1
print(var)
print(c1)
print(c2)
```

Training:

```
# Importing the Keras libraries and packages
from keras.models import Sequential
from keras.layers import Convolution2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense , Dropout
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "1"
sz = 128
# Step 1 - Building the CNN

# Initializing the CNN
classifier = Sequential()

# First convolution layer and pooling
classifier.add(Convolution2D(32, (3, 3), input_shape=(sz, sz, 1), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
# Second convolution layer and pooling
classifier.add(Convolution2D(32, (3, 3), activation='relu'))
# input_shape is going to be the pooled feature maps from the previous convolution layer
classifier.add(MaxPooling2D(pool_size=(2, 2)))
#classifier.add(Convolution2D(32, (3, 3), activation='relu'))
# input_shape is going to be the pooled feature maps from the previous convolution layer
#classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Flattening the layers
classifier.add(Flatten())

# Adding a fully connected layer
classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dropout(0.40))
classifier.add(Dense(units=96, activation='relu'))
classifier.add(Dropout(0.40))
classifier.add(Dense(units=64, activation='relu'))
classifier.add(Dense(units=27, activation='softmax')) # softmax for more than 2
```

```
# Compiling the CNN
classifier.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])


# Step 2 - Preparing the train/test data and training the model
classifier.summary()
# Code copied from - https://keras.io/preprocessing/image/
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory('data2/train',
                                                 target_size=(sz, sz),
                                                 batch_size=10,
                                                 color_mode='grayscale',
                                                 class_mode='categorical')

test_set = test_datagen.flow_from_directory('data2/test',
                                            target_size=(sz , sz),
                                            batch_size=10,
                                            color_mode='grayscale',
                                            class_mode='categorical')
classifier.fit_generator(
        training_set,
        steps_per_epoch=12841, # No of images in training set
        epochs=5,
        validation_data=test_set,
        validation_steps=4268)# No of images in test set
```

```
# Saving the model
model_json = classifier.to_json()
with open("model-bw.json", "w") as json_file:
    json_file.write(model_json)
print('Model Saved')
classifier.save_weights('model-bw.h5')
print('Weights saved')
```

App:

```
from PIL import Image, ImageTk
import tkinter as tk
import cv2
import os
import numpy as np
from keras.models import model_from_json
import operator
import time
import sys
import os
import matplotlib.pyplot as plt

#import cyhunspell

from spellchecker import SpellChecker
from string import ascii_uppercase
import pyttsx3
import os
language = 'en'
engine = pyttsx3.init()
class Application:
    def __init__(self):
        #self.hs = hunspell.HunSpell(
        #    '/usr/share/hunspell/en_US.dic', '/usr/share/hunspell/en_US.aff')
        dir(SpellChecker)

        spell=SpellChecker()
        self.hs=spell

        self.vs = cv2.VideoCapture(0)
        self.current_image = None
        self.current_image2 = None

        self.json_file = open("model/model-bw.json", "r")
        self.model_json = self.json_file.read()
```

```
        self.json_file.close()
        self.loaded_model = model_from_json(self.model_json)
        self.loaded_model.load_weights("model/model-bw.h5")

        self.json_file_dru = open("model/model-bw_dru.json", "r")
        self.model_json_dru = self.json_file_dru.read()
        self.json_file_dru.close()
        self.loaded_model_dru = model_from_json(self.model_json_dru)
        self.loaded_model_dru.load_weights("model/model-bw_dru.h5")

        self.json_file_tkdi = open("model/model-bw_tkdi.json", "r")
        self.model_json_tkdi = self.json_file_tkdi.read()
        self.json_file_tkdi.close()
        self.loaded_model_tkdi = model_from_json(self.model_json_tkdi)
        self.loaded_model_tkdi.load_weights("model/model-bw_tkdi.h5")

        self.json_file_smn = open("model/model-bw_smn.json", "r")
        self.model_json_smn = self.json_file_smn.read()
        self.json_file_smn.close()
        self.loaded_model_smn = model_from_json(self.model_json_smn)
        self.loaded_model_smn.load_weights("model/model-bw_smn.h5")

        self.ct = {}
        self.ct['blank'] = 0
        self.blank_flag = 0
        for i in ascii_uppercase:
            self.ct[i] = 0
        print("Loaded model from disk")
        self.root = tk.Tk()
        self.root.title("Sign language Detection")
        self.root.protocol('WM_DELETE_WINDOW', self.destructor)
        self.root.geometry("1500x1500")     #900x1100
        self.panel = tk.Label(self.root)
        self.panel.place(x=50, y=95, width=640, height=310)      # complete camera window
        self.panel2 = tk.Label(self.root)  # initialize image panel
```

```
        self.panel2.place(x=460, y=95, width=310, height=310)     # white window

        self.T = tk.Label(self.root)
        self.T.place(x=31, y=17)
        self.T.config(text="Sign Language to Text",
                      font=("courier", 40, "bold"))

        #def foo(self):
         #   self.img = ImageTk.PhotoImage(Image.open("signs.png"))     #Displaying Sign Language img
          #  self.canvas.create_image(nchor=NW, image=self.img)
           # self.canvas.image = self.img
            #self.img.place(x=800, y=500)

        self.photosign = tk.PhotoImage(file='pics/signs.png')
        self.w6 = tk.Label(self.root, image=self.photosign)
        self.w6.place(x=800, y=100)
        self.tx6 = tk.Label(self.root)
        self.tx6.place(x=800, y=17)
        self.tx6.config(text="Sign Languages",
                        font=("Courier", 30, "bold"))


        self.panel3 = tk.Label(self.root)  # Current Symbol
        self.panel3.place(x=500, y=480)    # 640 -> 480   # predicted symbol
        self.T1 = tk.Label(self.root)
        self.T1.place(x=75, y=480)         # Character 10 -> 70
        self.T1.config(text="Character :", font=("Courier", 40, "bold"))

        self.panel4 = tk.Label(self.root)  # Word
        self.panel4.place(x=350, y=550)     # 700 -> 550
        self.T2 = tk.Label(self.root)
        self.T2.place(x=80, y=550)          # 10 -> 80
        self.T2.config(text="Word :", font=("Courier", 40, "bold"))

        self.panel5 = tk.Label(self.root)  # Sentence
```

```
self.panel5.place(x=480, y=620)
self.T3 = tk.Label(self.root)
self.T3.place(x=85, y=620)
self.T3.config(text="Sentence :", font=("Courier", 40, "bold"))

#self.T4 = tk.Label(self.root)   # suggestions
#self.T4.place(x=400, y=690)     # x= 400 -> 900    y= 690 -> 480
#self.T4.config(text="", fg="red",
#              font=("Courier", 18))

#self.btcall = tk.Button(
   # self.root, command=self.action_call, height=0, width=0)
#self.btcall.config(text="About", font=("Courier", 14))
#self.btcall.place(x=825, y=0)

self.bt1 = tk.Button(
    self.root, command=self.action1, height=0, width=0)
self.bt1.place(x=26, y=890)
#self.bt1.grid(padx = 10, pady = 10)
self.bt2 = tk.Button(
    self.root, command=self.action2, height=0, width=0)
self.bt2.place(x=325, y=890)
#self.panel3.place(x = 10,y=660)
# self.bt2.grid(row = 4, column = 1, columnspan = 1, padx = 10, pady = 10, sticky = tk.NW)
self.bt3 = tk.Button(
    self.root, command=self.action3, height=0, width=0)
self.bt3.place(x=625, y=890)
# self.bt3.grid(row = 4, column = 2, columnspan = 1, padx = 10, pady = 10, sticky = tk.NW)
self.bt4 = tk.Button(
    self.root, command=self.action4, height=0, width=0)
self.bt4.place(x=125, y=950)
# self.bt4.grid(row = bt1, column = 0, columnspan = 1, padx = 10, pady = 10, sticky = tk.N)
self.bt5 = tk.Button(
    self.root, command=self.action5, height=0, width=0)
self.bt5.place(x=425, y=950)
```

```
    # self.bt5.grid(row = 5, column = 1, columnspan = 1, padx = 10, pady = 10, sticky = tk.N)
    self.str = ""
    self.word = ""
    self.current_symbol = "Empty"
    self.photo = "Empty"
    self.video_loop()

def video_loop(self):
    ok, frame = self.vs.read()
    if ok:
        cv2image = cv2.flip(frame, 1)
        x1 = int(0.5*frame.shape[1])
        y1 = 10
        x2 = frame.shape[1]-10
        y2 = int(0.5*frame.shape[1])
        cv2.rectangle(frame, (x1-1, y1-1), (x2+1, y2+1), (255, 0, 0), 1)
        cv2image = cv2.cvtColor(cv2image, cv2.COLOR_BGR2RGBA)
        self.current_image = Image.fromarray(cv2image)
        imgtk = ImageTk.PhotoImage(image=self.current_image)
        self.panel.imgtk = imgtk
        self.panel.config(image=imgtk)
        cv2image = cv2image[y1:y2, x1:x2]
        gray = cv2.cvtColor(cv2image, cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray, (5, 5), 2)
        th3 = cv2.adaptiveThreshold(
            blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 11, 2)
        ret, res = cv2.threshold(
            th3, 70, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
        self.predict(res)
        self.current_image2 = Image.fromarray(res)
        imgtk = ImageTk.PhotoImage(image=self.current_image2)
        self.panel2.imgtk = imgtk
        self.panel2.config(image=imgtk)
        self.panel3.config(text=self.current_symbol, font=("Courier", 50))  # Display font size
        self.panel4.config(text=self.word, font=("Courier", 40))
```

```
            self.panel5.config(text=self.str, font=("Courier", 40))

            if self.str !="":
                engine.say(self.str)
                engine.runAndWait()
            predicts = self.hs.correction(self.word)
            if(len(predicts) > 0):
                self.bt1.config(text=predicts[0], font=("Courier", 20))
            else:
                self.bt1.config(text="")
            if(len(predicts) > 1):
                self.bt2.config(text=predicts[1], font=("Courier", 20))
            else:
                self.bt2.config(text="")
            if(len(predicts) > 2):
                self.bt3.config(text=predicts[2], font=("Courier", 20))
            else:
                self.bt3.config(text="")
            if(len(predicts) > 3):
                self.bt4.config(text=predicts[3], font=("Courier", 20))
            else:
                self.bt4.config(text="")
            if(len(predicts) > 4):
                self.bt4.config(text=predicts[4], font=("Courier", 20))
            else:
                self.bt4.config(text="")
        self.root.after(30, self.video_loop)

    def predict(self, test_image):
        test_image = cv2.resize(test_image, (128, 128))
        result = self.loaded_model.predict(test_image.reshape(1, 128, 128, 1))
        result_dru = self.loaded_model_dru.predict(
            test_image.reshape(1, 128, 128, 1))
        result_tkdi = self.loaded_model_tkdi.predict(
            test_image.reshape(1, 128, 128, 1))
```

```
    result_smn = self.loaded_model_smn.predict(
        test_image.reshape(1, 128, 128, 1))
    prediction = {}
    prediction['blank'] = result[0][0]
    inde = 1
    for i in ascii_uppercase:
        prediction[i] = result[0][inde]
        inde += 1
    # LAYER 1
    prediction = sorted(prediction.items(),
                        key=operator.itemgetter(1), reverse=True)
    self.current_symbol = prediction[0][0]
    # LAYER 2
    if(self.current_symbol == 'D' or self.current_symbol == 'R' or self.current_symbol == 'U'):
        prediction = {}
        prediction['D'] = result_dru[0][0]
        prediction['R'] = result_dru[0][1]
        prediction['U'] = result_dru[0][2]
        prediction = sorted(prediction.items(),
                            key=operator.itemgetter(1), reverse=True)
        self.current_symbol = prediction[0][0]

    if(self.current_symbol == 'D' or self.current_symbol == 'I' or self.current_symbol == 'K' or self.current_symbol == 'T'):
        prediction = {}
        prediction['D'] = result_tkdi[0][0]
        prediction['I'] = result_tkdi[0][1]
        prediction['K'] = result_tkdi[0][2]
        prediction['T'] = result_tkdi[0][3]
        prediction = sorted(prediction.items(),
                            key=operator.itemgetter(1), reverse=True)
        self.current_symbol = prediction[0][0]

    if(self.current_symbol == 'M' or self.current_symbol == 'N' or self.current_symbol == 'S'):
        prediction1 = {}
        prediction1['M'] = result_smn[0][0]
```

```
        prediction1['N'] = result_smn[0][1]
        prediction1['S'] = result_smn[0][2]
        prediction1 = sorted(prediction1.items(),
                             key=operator.itemgetter(1), reverse=True)
        if(prediction1[0][0] == 'S'):
            self.current_symbol = prediction1[0][0]
        else:
            self.current_symbol = prediction[0][0]
    if(self.current_symbol == 'blank'):
        for i in ascii_uppercase:
            self.ct[i] = 0
    self.ct[self.current_symbol] += 1
    if(self.ct[self.current_symbol] > 60):
        for i in ascii_uppercase:
            if i == self.current_symbol:
                continue
            tmp = self.ct[self.current_symbol] - self.ct[i]
            if tmp < 0:
                tmp *= -1
            if tmp <= 20:
                self.ct['blank'] = 0
                for i in ascii_uppercase:
                    self.ct[i] = 0
                return
        self.ct['blank'] = 0
        for i in ascii_uppercase:
            self.ct[i] = 0
        if self.current_symbol == 'blank':
            if self.blank_flag == 0:
                self.blank_flag = 1
                if len(self.str) > 0:
                    self.str += " "
                self.str += self.word
                self.word = ""
```

```
            else:
                if(len(self.str) > 16):
                    self.str = ""
                self.blank_flag = 0
                self.word += self.current_symbol

    def action1(self):
        predicts = self.hs.suggest(self.word)
        if(len(predicts) > 0):
            self.word = ""
            self.str += " "
            self.str += predicts[0]

    def action2(self):
        predicts = self.hs.suggest(self.word)
        if(len(predicts) > 1):
            self.word = ""
            self.str += " "
            self.str += predicts[1]

    def action3(self):
        predicts = self.hs.suggest(self.word)
        if(len(predicts) > 2):
            self.word = ""
            self.str += " "
            self.str += predicts[2]

    def action4(self):
        predicts = self.hs.suggest(self.word)
        if(len(predicts) > 3):
            self.word = ""
            self.str += " "
            self.str += predicts[3]
```

```
    def action5(self):
        predicts = self.hs.suggest(self.word)
        if(len(predicts) > 4):
            self.word = ""
            self.str += " "
            self.str += predicts[4]

    def destructor(self):
        print("Closing Application...")
        self.root.destroy()
        self.vs.release()
        cv2.destroyAllWindows()

    def destructor1(self):
        print("Closing Application...")
        self.root.destroy()

print("Starting Application...")
pba = Application()
pba.root.mainloop()
```

CHAPTER 8
TESTING AND RESULTS

CHAPTER 8
TESTING AND RESULTS

The application will translate sign language to English and can be used by anybody.
The final product will be a complete application that will read the hand gestures from the user and convert it into English and give the output in audio format. Hence, our project shall serve as a very useful means for deaf-dumb people to communicate with others.
The unrecognized hand gestures are predicted with maximum probability.

For example, the gesture for alphabet ‘Y’ is shown according to the adjoining chart, on holding the gesture for few seconds within the Gaussian filter the alphabet is predicted as follows:

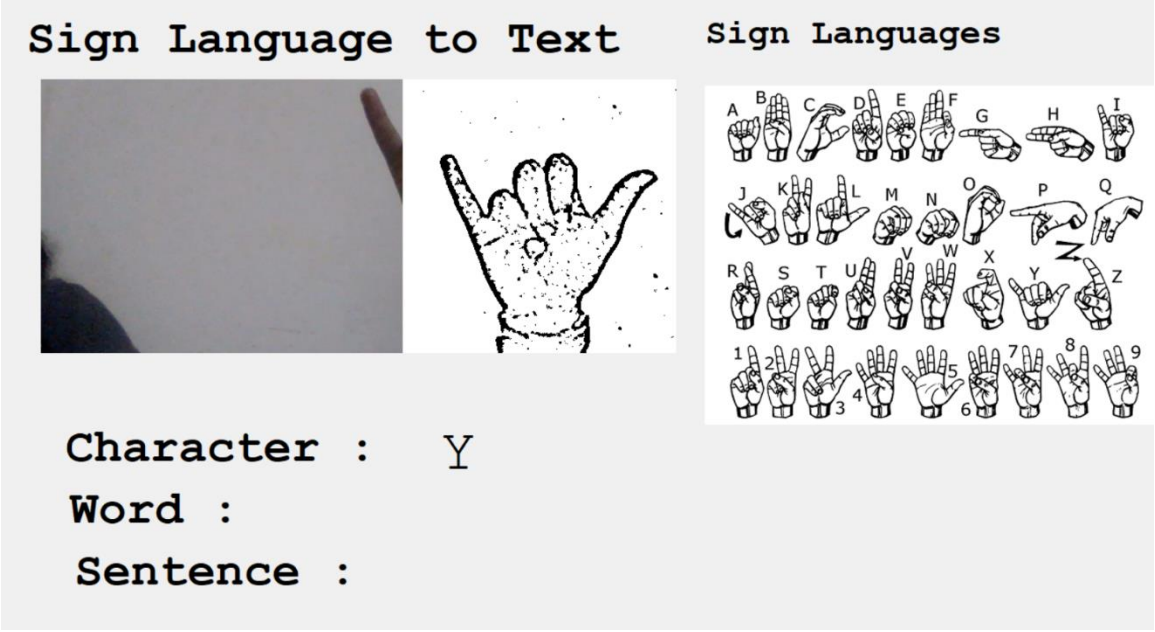


Figure 8.1: Output for alphabet ‘Y’

On leaving a gap of five seconds, a word is formed as follows:

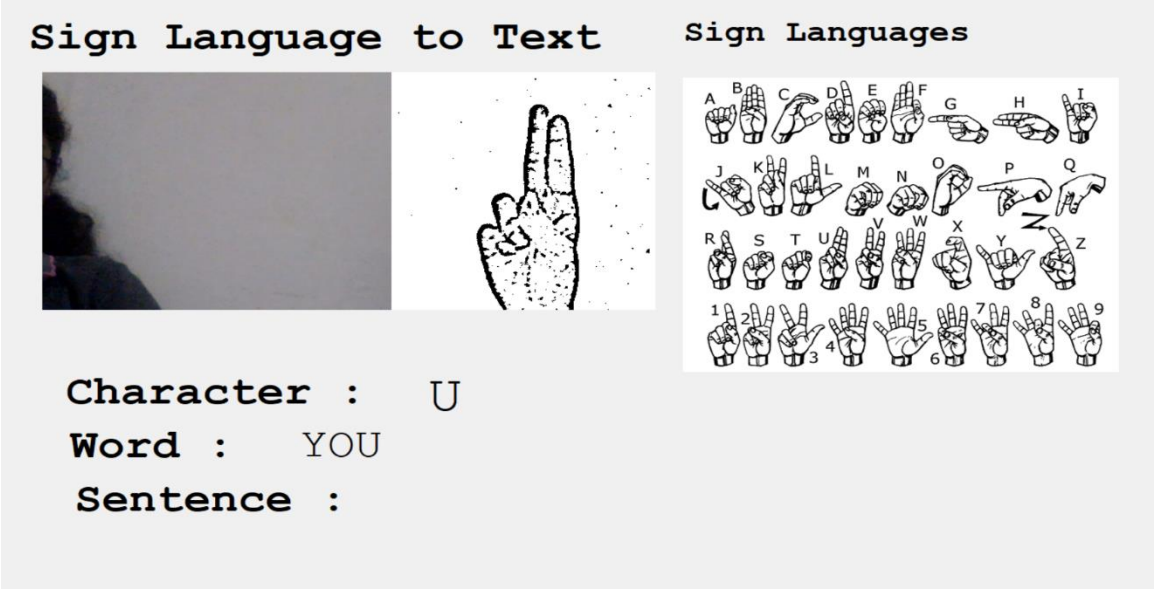


Figure 8.2: Output for word YOU

In order to give space between two words the screen should be left blank for five seconds:

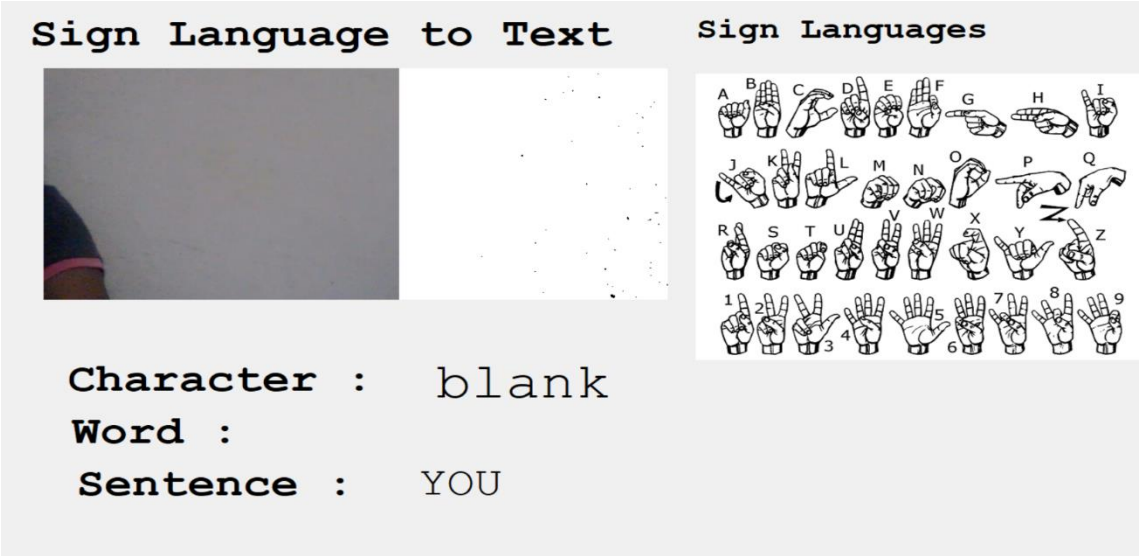


Figure 8.3: Output for a space between two words

The gesture for alphabet ‘D’ is shown according to the adjoining chart, on holding the gesture for few seconds within the Gaussian filter the alphabet is predicted as follows:

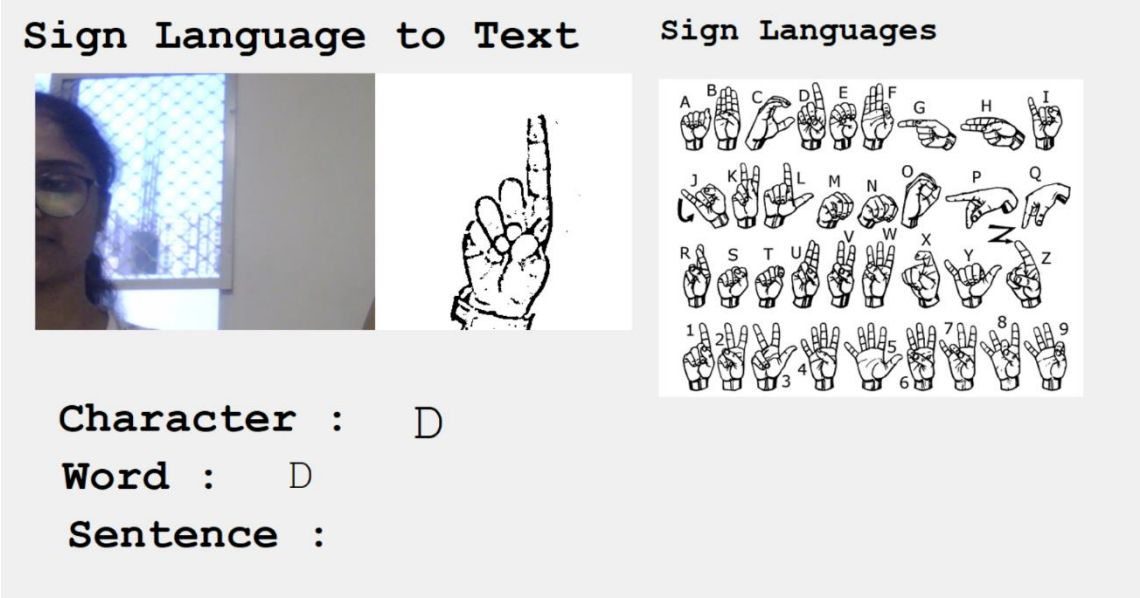


Figure 8.4: Output for alphabet 'D'

The gesture for alphabet 'K' is shown according to the adjoining chart, on holding the gesture for few seconds within the Gaussian filter the alphabet is predicted as follows:

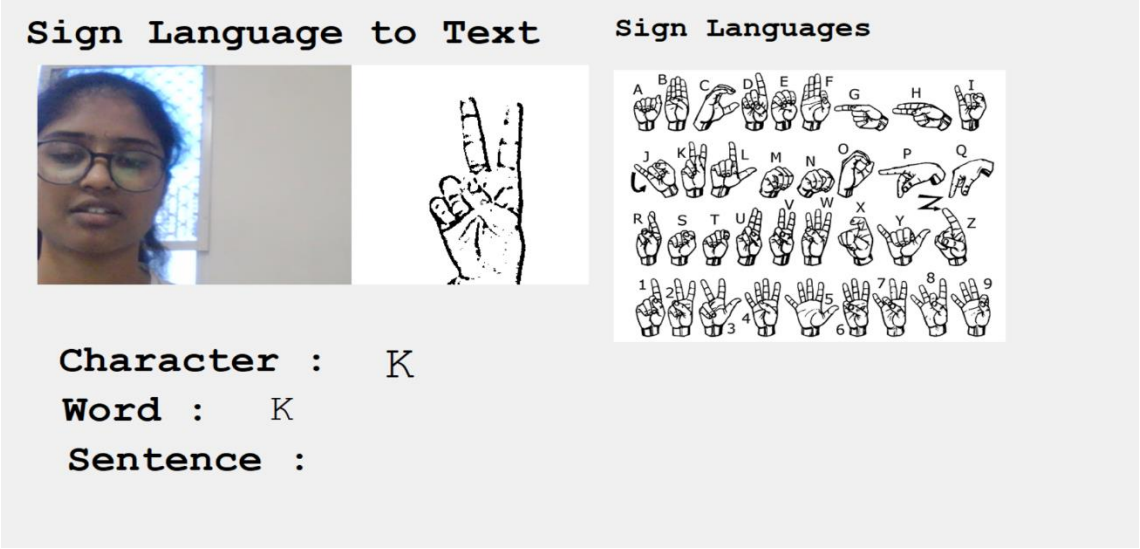


Figure 8.5: Output for alphabet 'K'

The gesture for alphabet 'L' is shown according to the adjoining chart, on holding the gesture for few seconds within the Gaussian filter the alphabet is predicted as follows:

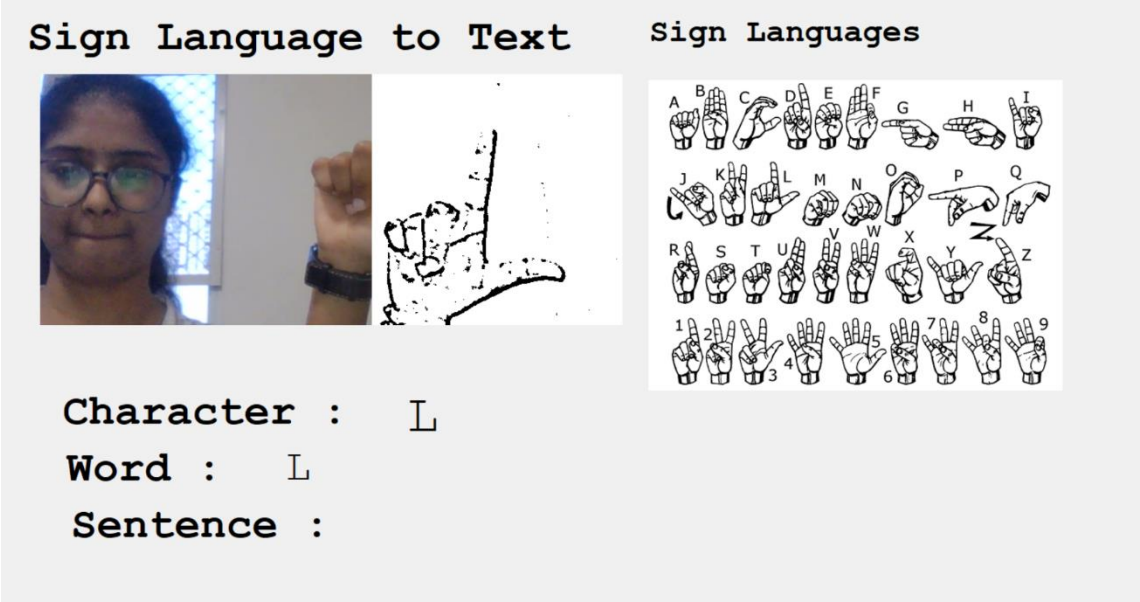


Figure 8.6: Output for alphabet 'L'

The gesture for alphabet 'P' is shown according to the adjoining chart, on holding the gesture for few seconds within the Gaussian filter the alphabet is predicted as follows:

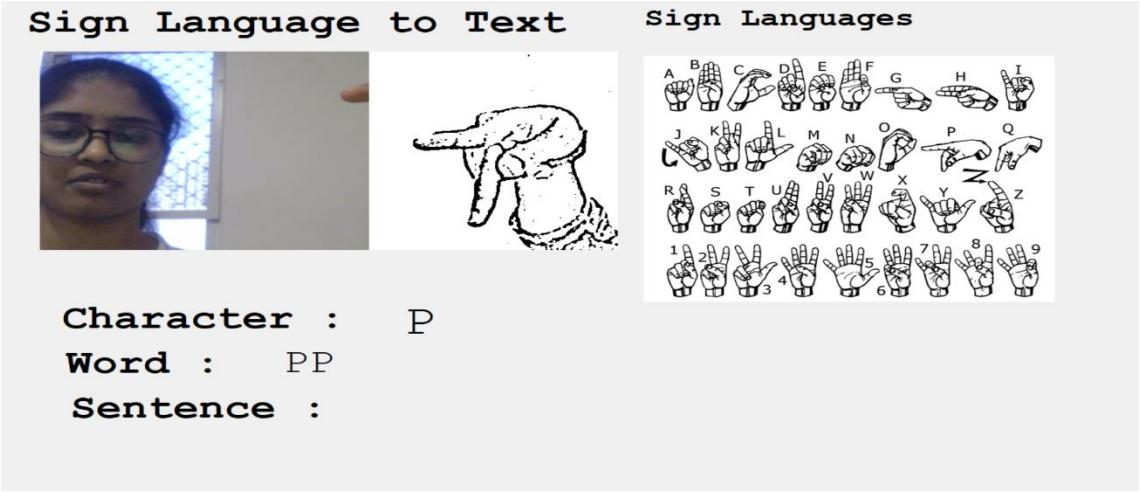


Figure 8.7: Output for alphabet 'P'

The gesture for alphabet 'R' is shown according to the adjoining chart, on holding the gesture for few seconds within the Gaussian filter the alphabet is predicted as follows:

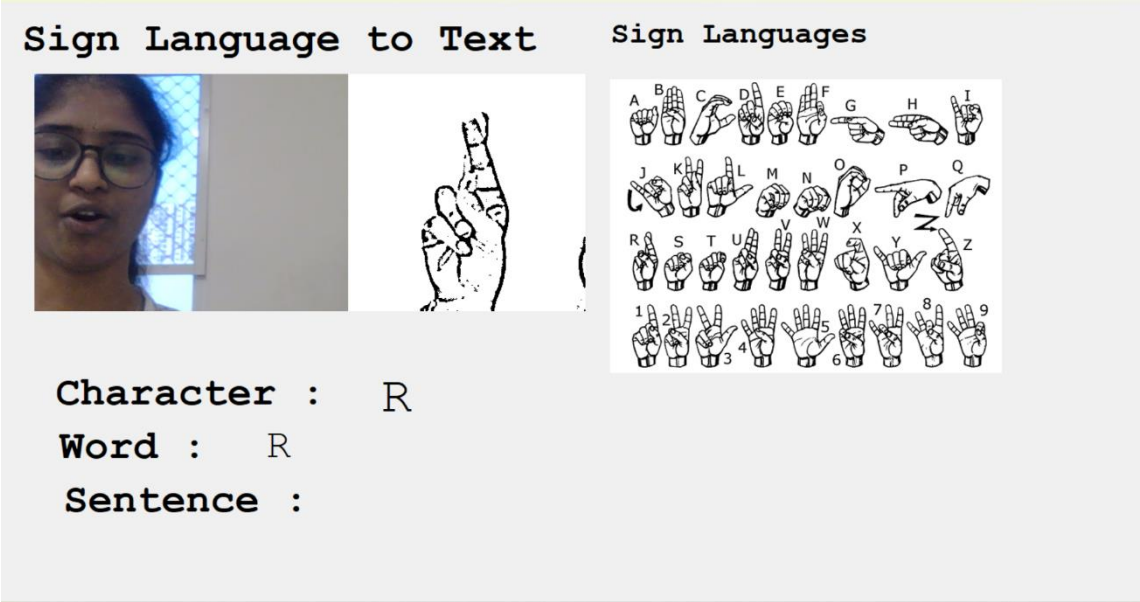


Figure 8.8: Output for alphabet 'R'

Result for Text to Audio:

With the help of OpenCV the images representing the gestures will be captured and text for the symbols shown will be generated, audio will also be generated for the same with the help of Engine.say function and Pyttsx3 module. Python Text to speech Version 3 (Pyttsx3) is a text-to-speech conversion library in Python. The pyttsx3 library is an extremely popular and highly-recommended Text-to-Speech (TTS) conversion library. It is fully supported by many popular operating systems and works offline with no delay. You can install pyttsx3 using the pip package manager. Once installed, pyttsx3 will load the right driver for your operating system. This includes sapi5 on Windows and espeak on Linux. Since it is compatible with any platform, you can use it with any TTS device.

CHAPTER 9

CONCLUSION AND FUTURE WORK

CHAPTER 9
CONCLUSION AND FUTURE WORK

9.1 CONCLUSION

Sign languages are kinds of visual languages that employ movements of hands, body, and facial expression as a means of communication. Sign languages are important for specially-abled people to have a means of communication. Through it, they can communicate and express and share their feelings with others. The drawback is that not everyone possesses the knowledge of sign languages which limits communication. This limitation can be overcome by the use of automated Sign Language Recognition systems which will be able to easily translate the sign language gestures into commonly spoken language. In this project, a real time computer vision based that uses American Sign Language recognition for the deaf-mute people have been created for the ASL alphabets. We have achieved an accuracy of 92% for training the dataset. Whereas, the manual testing is estimated to be 65%.On using Convolutional Neural Network, the accuracy of the application has increased considerably. The main motive to create the project was to take elementary steps towards breaking the barrier in communication between the deaf and dumb community and the normal people using the sign language.

9.2 FUTURE WORK

In a real-time situation, video processing is to be done, as the specially abled person's video needs to be captured and processed in frames for feature extraction and gesture identification. The system can be developed further to be compatible with various backgrounds, gesture placement with respect to the face position, hand movements, video jitters etc. And processing time needs to be minimized. An app can be developed, featured with this gesture recognition software, in order to communicate with doctors for easy conveyance of symptoms. This system can be further expanded into wireless communication through mobile phones by text to speech conversion and vice-versa. In the future, the dataset can be enlarged so that the system can recognize more gestures. The TensorFlow model that has been used can be interchanged with another model as well. The system can be implemented for different sign languages by changing the dataset.

REFERENCES

[1]Munir Oudah, Ali Al-Naji, Javaan Chahl, "Hand Gesture Recognition Based on Computer Vision: A Review of Techniques", MDPI, 2020.

[2]Abdullah Mujahid, Awais Yasin et al, "Real-Time Hand Gesture Recognition Based on Deep Learning", MDPI, 2021.

[3]Mais Yasen, Shaidah Jusoh, "A systematic review on hand gesture recognition techniques,challenges and applications", PeerJ 2019.

[4]Ghali Upendra, Kokkilig adda, D Gayathri, "Sign Language Interpreter using CNN", IJERT, 2021.

[5]K. Manikandan, Ayush Patidar, Pallav Walia, Aneek Barman Roy, “Hand Gesture Detection and Conversion to Speech and Text", arxiv, 2018.

[6]Amrita Thakur, Pujan Budhathoki, Sarmila Upreti, Shirish Shrestha, Subarna Shakya, "Real Time Sign Language and Speech Generation", JIIP, 2020.

[7]R Rumana, Reddygari Sandhya Rani, Mrs.R.Prema, "Sign Language Recognition for the deaf and dumb", IJERT, 2021.

[8]Sharvani Srivastava, Amisha Gangwar, Richa Mishra, Sudhakar Singh, "Sign Language Recognition System using TensorFlow Object Detection API", arxiv.

[9]Prashant Verma, Khushboo Badli, "Real-Time Sign Language Detection using TensorFlow, OpenCV and Python",ijraset, 2022.

[10]Poorvi Desai, Preksha Mohre, Soumya Bani, Varada S Naik, Sudha P.R, "Conversion of sign language to text for deaf and dumb", International Journal of Engineering Science Invention.

[11]Lakshman Karthik Ramkumar, Sudharsana Premchand, Gokul Karthi Vijayakumar, "Sign Language Recognition using Depth Data and CNN", SSRG International Journal of Computer Science and Engineering, 2019.

[12]Lohith DS, Nitin Raj, "Sign Language Recognition using Hand Gestures", 2021

[13]Akshay Goel, Raksha Tandon, Mandeep Singh Narula, "Sign Recognition and Speech Translation Using OpenCV", IRJET, 2020.

[14]Manasi Malge, Vidhi Deshmukh, Prof.Harshwardhan Kharpate, "Indian Sign Language Recognition", 2022.

[15]Dr. Pallavi Chaudhari, Pranay Pathrabe, Umang Ghatbandhe, Sangita Mondal, Sejal Parmar, ”Sign Language Detection System”, International Journal of Engineering Technology and Management Sciences, 2022.

[16]Shruti Chavan, Xinrui Yu, Jafar Saniie, “Convolutional Neural Network Hand Gesture Recognition for American Sign Language”.

[17]Shaheen Tabassum, Raghavendra R,”Sign Language Recognition and Converting into Text: A Survey”, JETIR, 2022.

[18]Sachin Devangan, Omkar Joshi, Shanu Jaiswal, Apratim Gholap, Netra Lokhande, “Vision-Based Hand Gesture Recognition Techniques using Smartphones”, International Journal of Innovative Technology and Exploring Engineering, 2022.

Github Link

https://github.com/CSE DSU/Team90_SignLanguageToSpeech