A

Mini Project Report

On

Project of Artificial Intelligence for Deaf and Aphonic

Submitted in partial fulfillment of the requirement for the award of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

OF

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY- HYDERABAD

By

Thogaruchesti Hemanth

205U1A05D2

Under the guidance

of

DR. Shaik Abdul Nabi

Vice Principal, HOD

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

AVN INSTITUTE OF ENGINEERING AND TECHNOLOGY

Approved by AICTE, Accredited by NAAC & NBA, Affiliated to JNTUH

Koheda Road, M.P. Patelguda Post, Ibrahimpatnam (M), Ranga Reddy Dist– 501 510. T.S. India.

2023-24

AVN INSTITUTE OF ENGINEERING & TECHNOLOGY
Accredited by NAAC & NBA

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEEING

CERTIFICATE

*This is to certify that the seminar report entitled "**PROJECT OF ARTIFICIAL INTELLIGENCE FOR DEAF AND APHONIC**" is a bonafide work done by "**THOGARUCHESTI HEMANTH** (205U1A05D2)" in the partial fulfillment of Bachelor of Technology in Computer Science and Engineering from JNTU, Hyderabad during the year 2020-24.*

Guide	Project Coordinator	HoD
Dr Shaik Abdul Nabi	Mr. Raju M. Tech	Dr Shaik Abdul Nabi
Vice Principal, HOD	Assistant Professor	Vice Principal, HOD

External Examiner

ACKNOWLEDGEMENT

I am greatly thankful to the following people for their contribution, advice, and assistance towards the development of this project and also in the completion of the report entitled "***Project of Artificial Intelligence for Deaf and Aphonic***".

I express my sincere thanks to My Coordinator **Mr. Raju,** Assistant Professor for the guidance, He showed me and for his sincere effort in building the seminar in the right direction.

I am indebted to **Dr**. **Shaik Abdul Nabi**, Head of the Department of CSE for his valuable suggestion and for the motivation he provided. My sincere thanks to him for sparing his valuable time and constant encouragement.

I would like to thank the Computer Science Department for giving me the opportunity to work on this project. This was quite a great experience and I learned a lot from it. It helped me to explore my skills and increased my interest in this project.

Specially thanks to **Mr. V Basha sir**, Assistant Professor for his invaluable guidance and support throughout this project, enhancing its efficiency and quality.

Thogaruchesti Hemanth

205U1A05D2

DECLARATION

I declare that the Project report entitled “**Project of Artificial Intelligence for Deaf and Aphonic** “recorded in this report does not form part of any other thesis on which a degree has been awarded earlier. I, further declare that this project report is based on my work carried out at the “**AVN Institute of Engineering and Technology**” in the I Semester of our final year B. Tech course.

Date: Signature

Place Thogaruchesti Hemanth

ABSTRACT

The deaf and the Aphonic, rely on some sort of sign language for developed and being used by those people for basic communication. Through this project we will be developing an application where a normal person can communicate with a deaf by giving their normal voice.in turn convert that to their respective sign language and produces the sign. Through this application the deaf will be able to find out what the normal person and our application is also useful for aphonic people they talk to other by hand sign. The signs will convert to their respective voice nodes. Through this application the Aphonic people also can communicate through their signs which will convert into voice. Gestures, such as voice-to-text conversion by using the google recognition. The text converted into Gestures Using deep learning Convolution Neural Network to train hand gesture.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF IMAGES

CHAPTER 1
INTRODUCTION

1.1 Introduction

American sign language is a predominant sign language Since the only disability Deaf and aphonic people have been communication related and since they cannot use spoken languages, the only way for them to communicate is through sign language. Aphonic is a condition or state characterized by the absence or loss of voice or sound. It is often used to describe a person who is unable to produce vocal sounds or speech. Communication is the process of exchange of thoughts and messages in various ways such as speech, signals, behavior and visuals. Deaf and Aphonic people make use of their hands to express different gestures to express their ideas with other people. Gestures are the non-verbally exchanged messages and these gestures are understood with vision. This nonverbal communication of deaf and Aphonic people is called sign language. Contrary to popular belief, sign language is not international. These vary from region to region.

Sign language is a visual language and consists of 3 major components:

Fingerspelling	**Word level sign vocabulary**	**Non-manual features**
Used to spell words letter by letter.	Used for the majority of communication	Facial expressions and tongue, mouth and body position.

Table 1.1: -Types of Deaf and Aphonic people communication

In our project we primarily focus on producing a model which can recognize Fingerspelling based hand gestures in order to form a complete word by combining each gesture. The gestures we aim to train are as given in the image below.

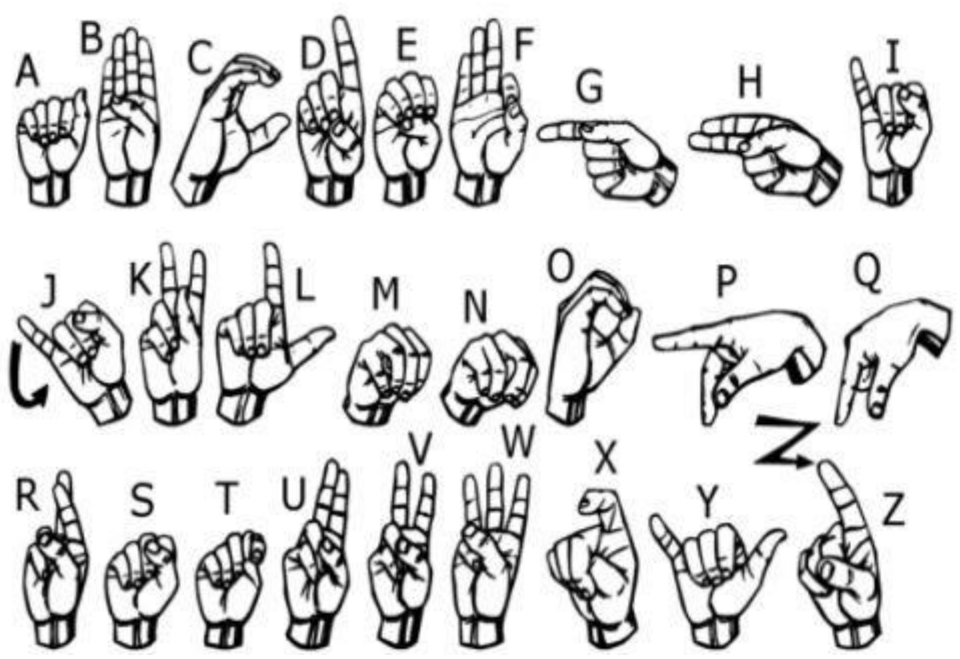

Figure-1.1 Alphabet signs in American Sign Language

1.2 Existing System

Vision-Based Approach: The vision-based approach relies on cameras to capture and interpret gesture images. Through image processing techniques and machine learning algorithms, key features within these images are extracted for gesture recognition. This approach boasts high accuracy and robustness in recognizing gestures, allowing for natural and intuitive interactions.

Limitations

- Processing overhead: The computational requirements for real-time image processing can be demanding, affecting system responsiveness.

Sensor-Based Approach: The sensor-based approach involves utilizing various sensors placed on the hand to record data during gesture execution. Analyzing this data assists in recognizing gestures, but it comes with limitations. The use of external hardware can interfere with natural hand movements, potentially disrupting the user experience. Additionally, recognizing complex gestures might pose challenges for this method.

Limitations

- Complexity in recognizing nuanced gestures: Complex or intricate gestures might pose challenges in accurate recognition due to limitations in sensor data interpretation.

1.3 Proposed System

The proposed system aims to develop a software application leveraging machine learning techniques to interpret and translate American Sign Language (ASL) alphabets. The system will employ a dataset comprising the 26 ASL alphabets to train a model capable of recognizing and converting textual input or audio format into video representations of hand gestures and corresponding audio output of the interpreted text.

Advantages over the Existing System:

1. **ASL Alphabet Translation:** The system will interpret input text or audio and generate corresponding hand gesture videos representing ASL alphabets.
2. **Audio Output:** Alongside visual representations, the system will produce sound for the respective interpreted text, enhancing accessibility for users.
3. **User-Friendly Interface:** An intuitive interface will allow users to input text or audio and receive both video and audio outputs effectively.

1.4 Problem Statement

The deaf and aphonic community encounter substantial communication barriers often overlooked in conventional methods. These hurdles, relying on spoken or text-based communication, impede effective interaction and learning for those dependent on sign language. Our initiative emerges from a profound desire to bridge this gap by leveraging artificial intelligence. We aim to empower this community by offering a tool converting text or audio input into sign language. This technology doesn't just facilitate communication but also serves as an educational resource, enhancing learning and fostering inclusivity.

1.5 Hardware Requirements

- **Processor:** i3 or higher for efficient computation.
- **RAM:** 8GB for smooth multitasking.
- **Graphics Card:** 4GB dedicated card for visuals.
- **Storage:** 255GB SSD for fast data access.
- **Camera:** Good-quality for clear imaging.
- **Microphone:** Quality support for audio clarity.

1.6 Software Requirements

- **Programming Language:** Python 3.6.6
- **Libraries and Frameworks:**
- ✓ **TensorFlow 1.11.0:** For machine learning model development, focusing on neural network architectures.
- ✓ **OpenCV 3.4.3.18:** Used for computer vision tasks, including image and video analysis, offering diverse functionalities like object detection.
- ✓ **NumPy 1.15.3:** Vital for numerical computations and array manipulations, crucial for data handling and mathematical operations.
- ✓ **Matplotlib 3.0.0:** Enables data visualization and plotting, facilitating graphical representation of data sets and results.
- ✓ **Keras 2.2.1:** High-level neural networks API, serving as an interface for TensorFlow, simplifying model building.
- ✓ **PIL (Python Imaging Library) 5.3.0**: Essential for image processing tasks like resizing and basic manipulations.
- **Additional Components:**
- ✓ **Tkinter:** Used for creating the graphical user interface (GUI) for user interaction.
- **Hardware Dependencies:**
- ✓ **CUDA 12.2 (GUI):** Enhances performance for computationally intensive tasks by enabling parallel computing on NVIDIA GPUs.

CHAPTER 2
LITERATURE SURVEY

TITLE: Sign Language to Speech Conversion Using Image Processing and Machine Learning

Author: Prof. Chhaya Narvekar, Sayukta Mungekar, Arpita Pandey, Mansi Mahadik

Extensive literature examines Sign Language recognition methodologies for the deaf and aphonic, focusing on Vision-Based and Sensor-Based approaches. Vision-Based methods use cameras to capture gestures, employing image processing for feature extraction and recognition, excelling in accuracy. These methods convert images to speech or text via machine learning. Conversely, Sensor-Based approaches use hand sensors, compromising natural movements but analyzing gesture data. Discussions often revolve around balancing accuracy and natural gesture complexity. Image Processing, like the Otsu Method for image thresholding, plays a vital role in separating foreground and background. Machine Learning algorithms, trained on data, autonomously predict and decide, crucial for effective sign language recognition. The integration of image processing with machine learning stands as a critical advancement in communication interfaces for the deaf and aphonic.

TITLE: Sign Language Recognition by Image Processing

Author: Sabari Priya A, Adrija Nair, Sreejin Madhavan, Kavitha Issac

Neural networks, especially Convolutional Neural Networks (CNNs), are modern AI marvels transforming industries. Developed from the concept of mimicking the brain's processing, they unravel complex data relationships. Yann LeCun's 1988 pioneering work made CNNs pivotal in image classification, heavily used by tech giants. These networks view images as pixel arrays, interpreting data through convolutional, nonlinear, and pooling layers. In a 300 x 300 image, RGB values (0 to 255) guide the network's understanding of pixel intensity. CNNs' intricate layers extract features, crucial for accurate outputs in tasks like tagging, recommendations, and photo searches.

TITLE: SIGNGRAPH: An Efficient and Accurate Pose-Based Graph Convolution Approach Toward Sign Language Recognition

Author: Neelma Naz, Hasan Sajid, Sara Ali, Osman Hasan

Sign Language Recognition (SLR) has evolved with parallels to gesture and action recognition, leveraging three main categories: Appearance-based, Pose/Skeleton-based, and Hybrid methods. Appearance-based methods focus on spatial and temporal features in videos, using CNNs for spatial features and RNNs for temporal information. Techniques like Single Shot Detectors (SSD) and LSTM models are employed to estimate hand features and capture sign language movements. Motion History Images (MHIs) and motion templates like RGB motion images, combined with models like I3D, extract spatiotemporal dependencies from video data.

Skeleton-based methods leverage advancements in pose estimation, extracting skeletal joint data with approaches like top-down or bottom-up techniques. These methods use GRUs, TGCNs, GCNs, BERT models, and transformers to process pose data, historically less accurate but more computationally efficient compared to appearance-based methods

Hybrid methods integrate various data modalities (RGB, depth, optical flow, skeleton) through fusion strategies, enhancing accuracy. Combining modalities like skeleton, optical flow, RGB, depth, and others into multi-modal ensembles significantly improves accuracy, albeit requiring substantial processing power. These methods involve manual and non-manual feature extraction followed by fusion using models like 2D-CNN-LSTM and 3D-CNN with pose-guided pooling mechanisms, showcasing their potential despite computational demands.

CHAPTHER 3
REQURIEMENT ANALYSIS

3.1 System Analysis

The system serves as a bridge between input text or audio and the representation of American Sign Language (ASL) through a database-driven approach. It encompasses preprocessing, data comparison, feature extraction, and output generation to facilitate accurate translation into ASL images.

Components:

1. Input Module:

- Accepts input from users through keyboard entry.
- Allows selection between text and audio inputs.

2. Preprocessing:

- Determines the type of input (text or audio) for subsequent processing.
- Ensures the correctness and appropriateness of the input for seamless processing.

3. Data Comparison

- Matches parameters obtained from input (text or audio) with the database.
- Uses comparison algorithms to find corresponding values.

4. Database

- Central repository containing:
- Images of alphabets in ASL.
- Commonly used words in ASL represented as images.
- Sentences in ASL represented through a sequence of images.
- Essential for accurate and effective output generation.

5. Feature Extraction

- Extracts features from input text.
- Matches these extracted features with stored database features.

6. Translation Module

- Converts text inputs into ASL images based on matched database entries.
- Generates sign language representations for:
- Individual alphabet characters.
- Frequently used words.

3.1.1 Feasibility Study

The project focuses on developing an application to facilitate seamless communication between sign language and spoken language users, primarily benefiting the deaf, aphonic, and mute communities. It utilizes Python as the primary programming language, employing TensorFlow, OpenCV, NumPy, and Keras for robust AI-driven functionalities. Hardware dependencies on CUDA for NVIDIA GPUs enhance computational efficiency, ensuring real-time interaction.

The user-friendly application aims to bridge language barriers, designed with Tkinter for accessibility across diverse user groups. Deep learning models like Convolutional Neural Networks enable swift and accurate gesture recognition and translation between sign and spoken languages. Though initial investment in technology and hardware is necessary, the long-term benefits and potential market impact, serving a significant demographic, make it economically viable through donations, grants, or commercialization.

3.2 Methodology and Modules

The system is a vision-based approach. All signs are represented with bare hands and so it eliminates the problem of using any artificial devices for interaction.

Data Set:

For the project we tried to find already made datasets, we find dataset in the form of raw images that matched our requirements. They are as how below

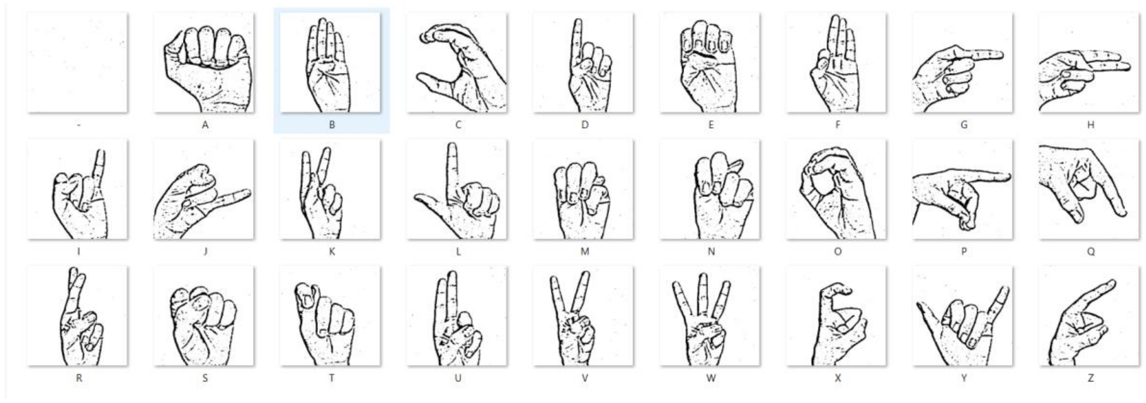

Figure – 3.1 Data Set Alphabet Signs for Training the Model

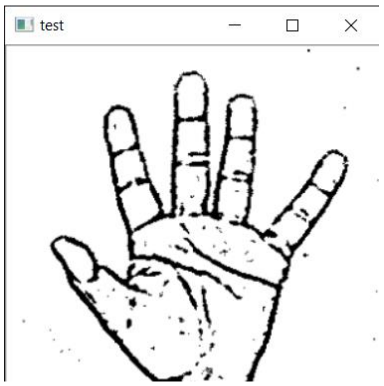

Figure – 3.2 Gaussian Blurred Image

Module 1: CNN Model Creation

This module encompasses the development of a Convolutional Neural Network (CNN) model leveraging Python programming. It focuses on utilizing deep learning methodologies to facilitate gesture recognition and translation between sign language and spoken language. The CNN model creation involves the integration of powerful libraries like TensorFlow, OpenCV, NumPy, and Keras. These tools enable the construction of a robust AI-driven system capable of accurately recognizing and interpreting gestures, forming a foundational aspect of the application.

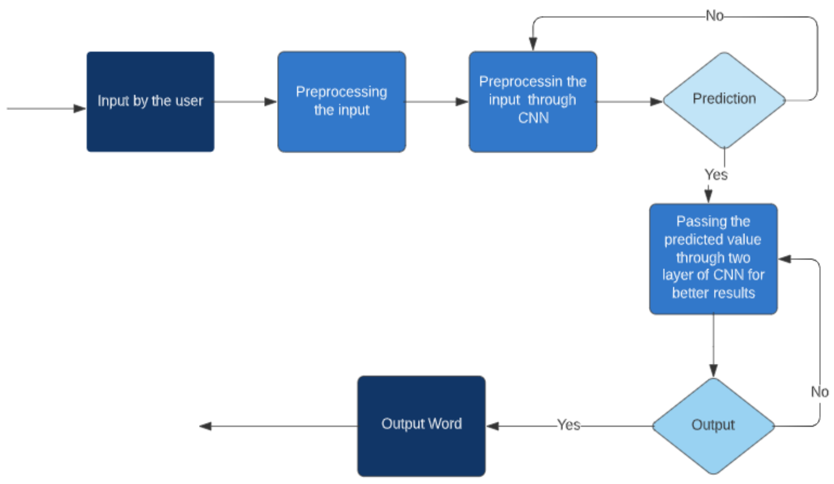
Input by the user
Preprocessing the input
Preprocessin the input through CNN
Prediction
No
Yes
Passing the predicted value through two layer of CNN for better results
Output
No
Yes
Output Word

Figure – 3.3 Architecture for Gesture Classification

Module 2: System Design for Deaf and Aphonic Communication

The second module involves the comprehensive design and development of a system catered specifically for the deaf and aphonic communities. It integrates various technologies such as CUDA for GPU acceleration, Tkinter for a user-friendly interface, and Python for system integration. CUDA plays a crucial role in enhancing computational efficiency, particularly for real-time processing necessary for gesture recognition and translation. Tkinter is employed to design an intuitive user interface ensuring accessibility and ease of use for diverse user groups. This module focuses on creating a seamless system that facilitates communication between sign language and spoken language users, addressing the unique needs of the deaf and aphonic individuals through a well-integrated and functional design.

3.3 System Design:

- **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 (ReLU):** Utilized ReLU (Rectified Linear Unit) for each layer, introducing nonlinearity and aiding in learning intricate features while addressing the vanishing gradient problem, accelerating training by reducing computation time.
- **Pooling Layer (Max Pooling):** Applied Max pooling with a (2, 2) pool size along with ReLU activation. This reduction technique cuts parameters, minimizing computational costs and mitigating overfitting.
- **Dropout Layers:** Employed to combat overfitting by randomly deactivating a subset of activations in a layer, ensuring the network's capability to classify accurately despite dropped activations.
- **Optimizer (Adam):** Implemented the Adam optimizer, leveraging the strengths of ADA GRAD and RMSProp, refining the model concerning the output of the loss function for effective updates.

3.3.1 Block Diagram /Architecture

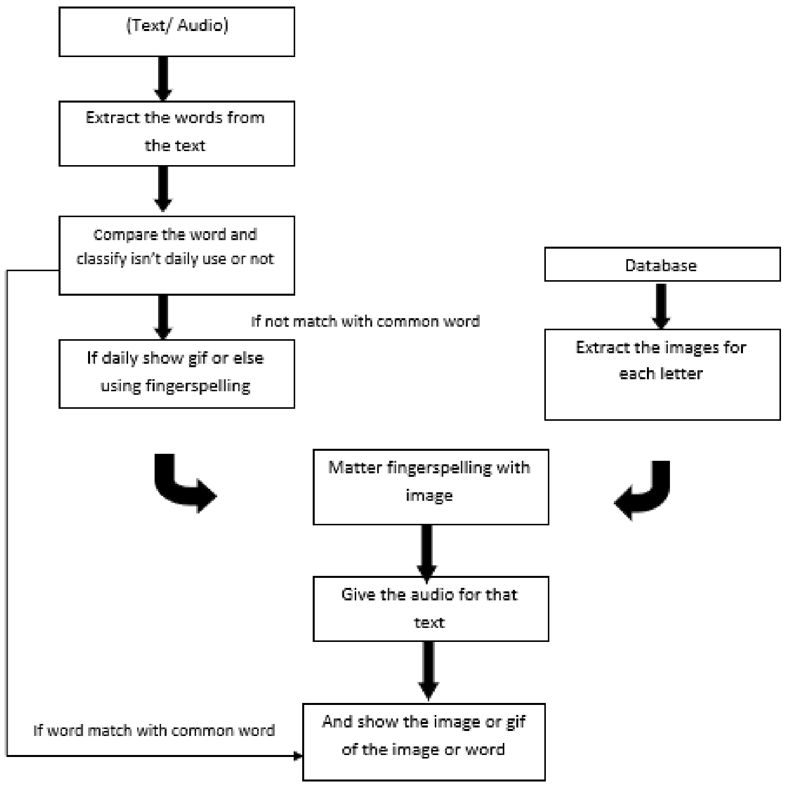
(Text/ Audio)
Extract the words from the text
Compare the word and classify isn't daily use or not
If not match with common word
If daily show gif or else using fingerspelling
Database
Extract the images for each letter
Matter fingerspelling with image
Give the audio for that text
If word match with common word
And show the image or gif of the image or word

Figure 3.4 Block Diagram of the System for the Deaf Option

The Block Diagram depicted in Figure 3.3.1 illustrates the systematic flow of data processing within the system, specifically when the "Deaf" option is selected. This diagram intricately showcases the sequential processing of input data, elucidating how the system handles and interprets information. It delineates the internal mechanisms orchestrating data processing, showcasing the intricate steps from input acquisition to output generation. The diagram comprehensively elucidates the inner workings, illustrating the transformation and interpretation of data to facilitate effective communication for individuals relying on sign language.

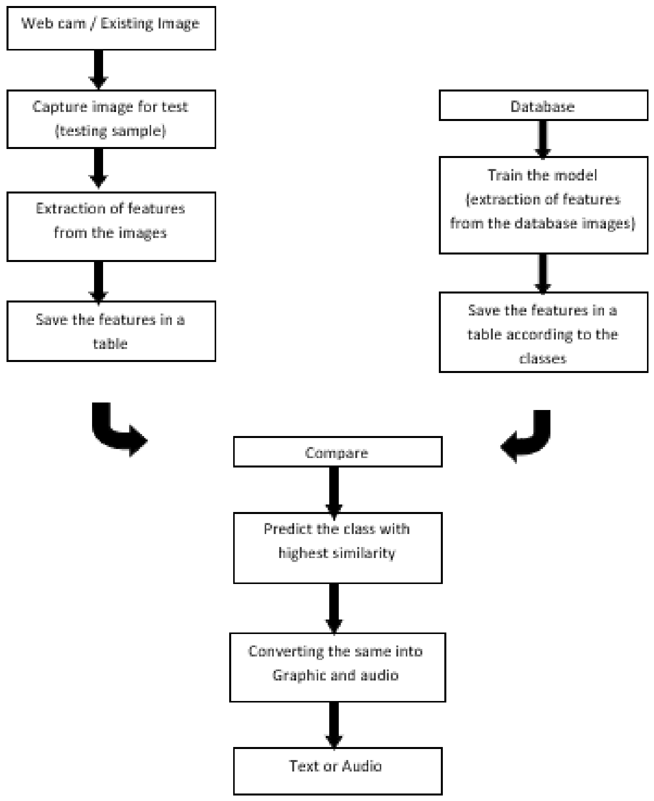
Web cam / Existing Image
Capture image for test (testing sample)
Extraction of features from the images
Save the features in a table
Database
Train the model (extraction of features from the database images)
Save the features in a table according to the classes
Compare
Predict the class with highest similarity
Converting the same into Graphic and audio
Text or Audio

Figure 3.5 Block Diagram of the System for the Aphonic People

The Block Diagram depicted in Figure 3.3.2 illustrates the system's flow following the selection of the Aphonic option. It delineates the user input acquisition, processing stages within the system, and the utilization of the dataset and trained model. The diagram outlines the sequence of events from input acquisition, where user input is received, to the subsequent processing steps within the system. It elucidates the significance of the dataset and trained model, showcasing how these elements contribute to the system's functionality. Additionally, the diagram delineates the actions taken when specific inputs are provided, detailing the subsequent processes within the system to enable effective communication for aphonic users.

3.3.2 UML Diagrams

UML is a method for describing the system architecture in detail using the blueprint. UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. UML is a very important part of developing Objects oriented software and the software development process. UML uses mostly graphical notations to express the design Of software projects. Using the UML helps project teams communicate, explore potential designs.

Definition:

UML is a general-purpose visual modeling language that is used to specify, visualize, construct, and document the artifacts of the software system.

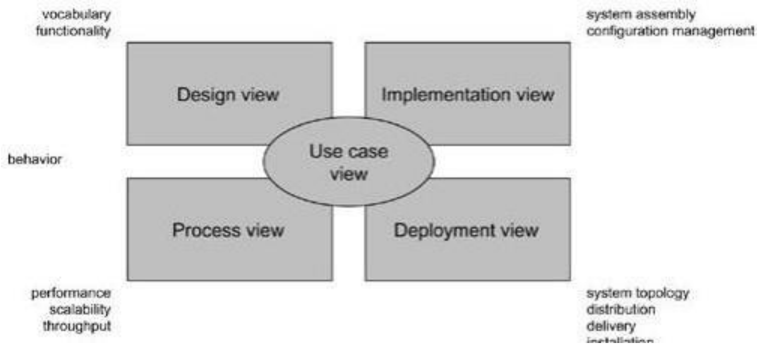

Figure 3.6 UML view

Use-Case diagram:

A use case is a set of scenarios that describing an interaction between a user and a system. A use case diagram displays the relationship among actors and use cases. The two main components of a use case diagram are use cases and actors.

Use case diagram Figure 3.7, the interaction between the user and system is illustrated explicitly in the scenario where the "Aphonic" option is selected. The system's functionality is defined to receive input in the form of an image, representing the user's communication, and subsequently processes this input. The system's operation involves a sophisticated conversion mechanism where the image input undergoes analysis and recognition processes. Result, the system generates two primary outputs: text and audio representations. This functionality is specifically tailored to cater to the needs of aphonic individuals, ensuring inclusivity and accessibility by providing text and audio outputs as a means of communication in response to the user's input image.

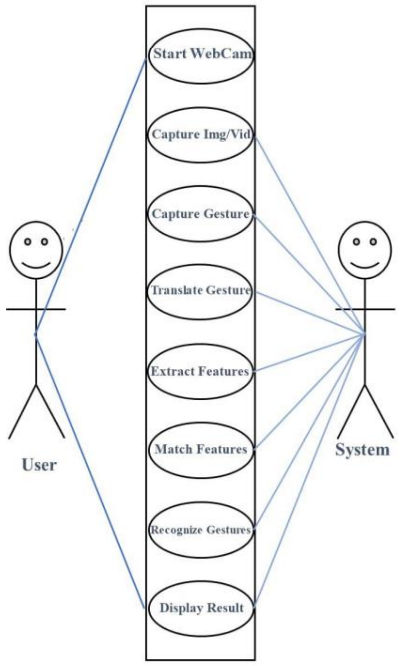

Figure 3.7 Use Case Diagram for Aphonic

Data Flow Diagram:

A data flow diagram (DFD) serves as a visual representation of how data moves through a system. It illustrates the flow of information, depicting the inputs, outputs, processes, and data storage within a system. These diagrams categorize data into different levels of abstraction, typically starting with a context diagram showing the system as a whole, followed by more detailed diagrams breaking down processes and interactions. DFDs facilitate a comprehensive understanding of a system's functionality, aiding in communication between stakeholders, identifying areas for improvement or optimization, and serving as a foundational tool for documentation, system design, and development processes. Through its hierarchical structure and clear depiction of data movement, a well-constructed DFD forms a crucial component of comprehensive system documentation, ensuring clarity and coherence in conveying complex system behaviors and interactions.

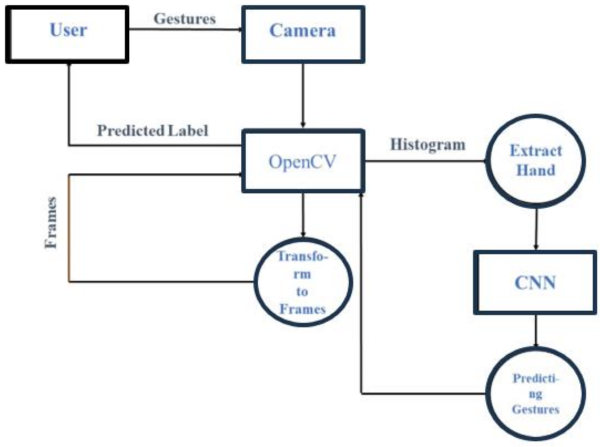

Figure 3.7 Data Flow Diagram

Sequence Diagram

A sequence diagram is a vital tool for visualizing interactions between various components or actors in a system. It illustrates the flow of messages or actions among these entities, showcasing the order in which they occur. Typically used in software development, these diagrams provide a clear depiction of the dynamic behavior within a system, helping developers, stakeholders, and designers understand the chronology of events and the dependencies between different elements. By detailing the sequence of interactions, message passing, and the temporal aspects of a system's functionalities, sequence diagrams serve as comprehensive documentation, aiding in system analysis, design, and troubleshooting, ultimately contributing to the development of robust and efficient systems.

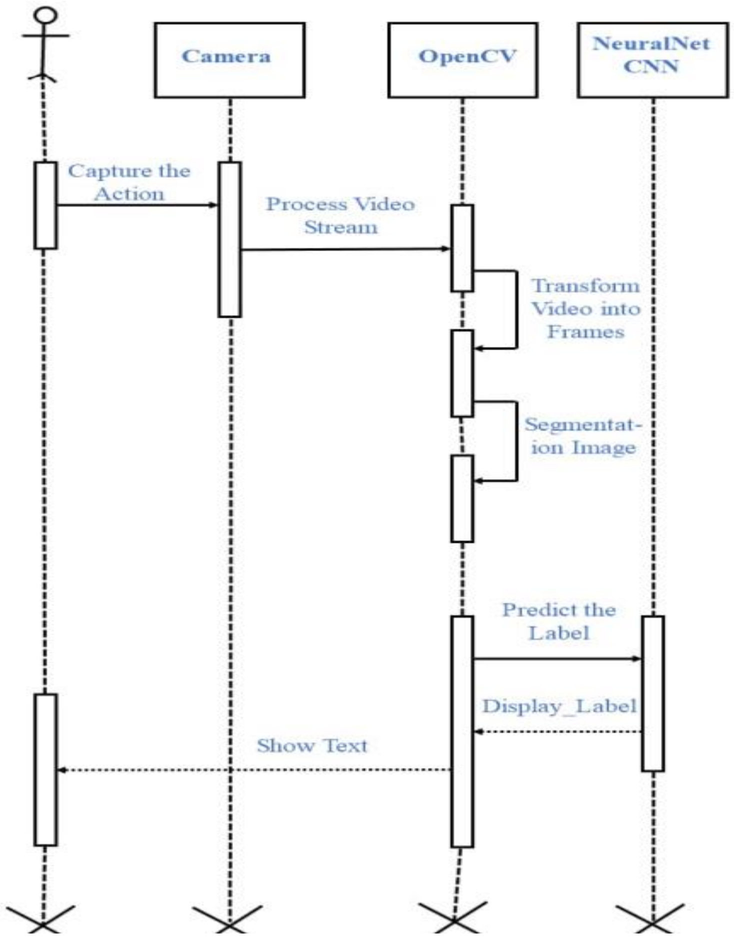

Figure 3.7 Deaf Sequence Diagram

CHAPTER 4
IMPLEMENTATION

4.1 Technologies

1. Artificial Neural Network (ANN):

Artificial Neural Network is a connection of neurons, replicating the structure of human brain. Each connection of neuron transfers information to another neuron. Inputs are fed into first layer of neurons which processes it and transfers to another layer of neurons called as hidden layers. After processing of information through multiple layers of hidden layers, information is passed to final output layer.

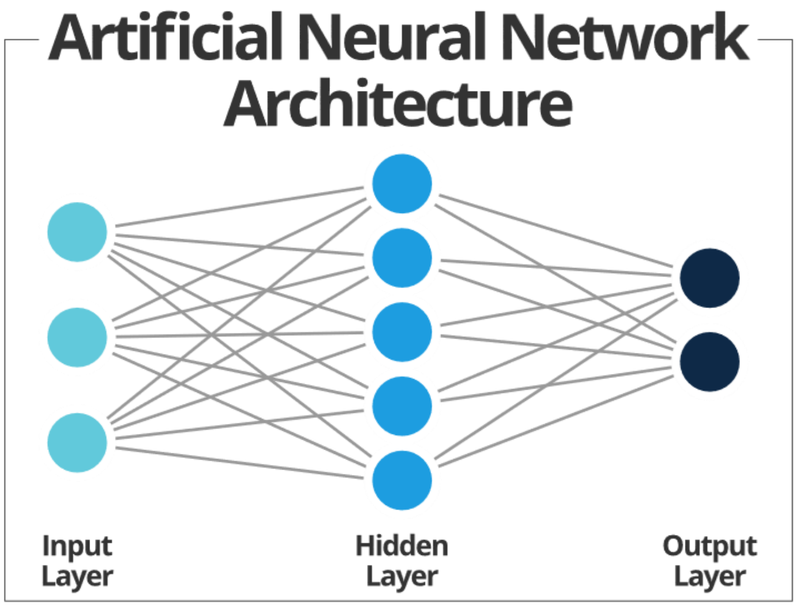

Figure – 4.1 Artificial Neural Network Architecture

These are capable of learning and have to be trained. There are different learning strategies:

1. Unsupervised Learning
2. Supervised Learning
3. Reinforcement Learning

2. Convolutional Neural Network (CNN):

Unlike regular Neural Networks, in the layers of CNN, the neurons are arranged in 3 dimensions: width, height, depth. The neurons in a layer will only be connected to a small region of the layer (window size) before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would have dimensions (number of classes), because by the end of the CNN architecture we will reduce the full image into a single vector of class scores.

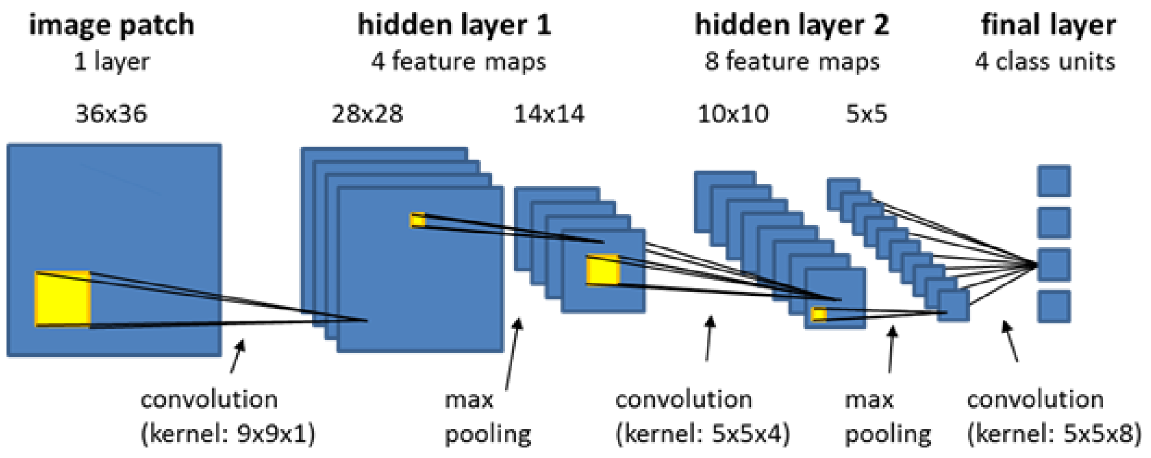

Figure – 4.2 CNN Model Architecture

i. **Convolution Layer:** In convolution layer we take a small window size [typically of length 5*5] that extends to the depth of the input matrix. The layer consists of learnable filters of window size. During every iteration we slid the window by stride size [typically 1], and compute the dot product of filter entries and input values at a given position.

ii. **Pooling Layer:** We use pooling layer to decrease the size of activation matrix and ultimately reduce the learnable parameters. There are two types of pooling:

a. Max Pooling: In max pooling we take a window size [for example window of size 2*2], and only take the maximum of 4 values. Well lid this window and continue this process, so well finally get an activation matrix half of its original Size.

b. Average Pooling:_ In average pooling, we take advantage of all Values in a window.

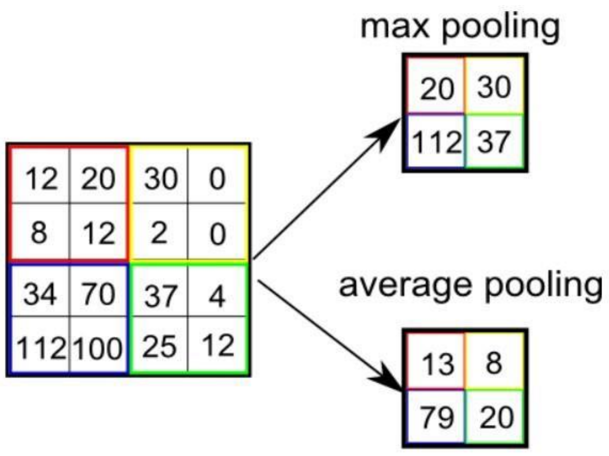

Figure – 4.3 Types of Pooling Layers

iii. Fully Connected Layer: In convolution layer, neurons are connected only to a local region, while in a fully connected region, we will connect all the inputs to neurons.

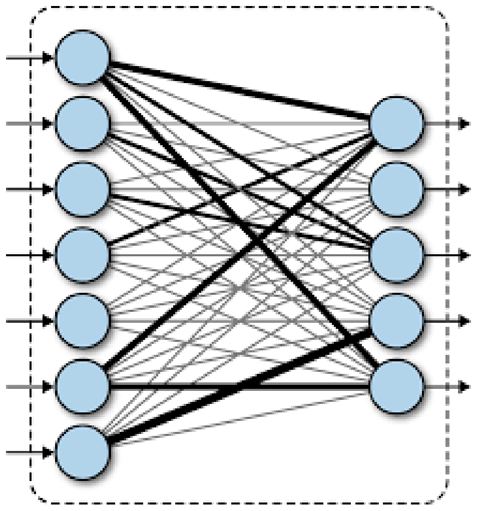

Figure – 4.4 Fully Connected Layer

iv. **Final Output Layer:** After getting values from fully connected layer, we will connect them to the final layer of neurons [having count equal to total number of classes], that will predict the probability of each image to be in different classes.

3. 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.

4. 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. It contains implementations of commonly used neural network elements like layers, objective, activation functions, optimizers, and tools to make working with images and text data easier.

5. OpenCV:

OpenCV (Open-Source Computer Vision) is an open-source library of programming functions used for real-time computer-vision. It is mainly used for image processing, video capture and analysis for features like face and object recognition. It is written in C++ which is its primary interface, however bindings are available for Python, Java, MATLAB. The system is a vision-based approach. All signs are represented with bare hands and so it eliminates the problem of using any artificial devices for interaction.

4.2 Code Implementation

Code for CNN Model:

```
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
import os
# Set visible GPU device
os.environ["CUDA_VISIBLE_DEVICES"] = "1"
```

```
# Define data generators
train_datagen = ImageDataGenerator( rescale=1./255,shear_range=0.2,zoom_range=0.2,
horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)
# Load training and testing data
training_set = train_datagen.flow_from_directory(
    'D:\\PANDA\\dataSet\\trainingData',
    target_size=(128, 128),
    batch_size=10,
    color_mode='grayscale',
    class_mode='categorical'
)
test_set = test_datagen.flow_from_directory(
    'D:\\PANDA\\dataSet\\testingData',
    target_size=(128, 128),
    batch_size=10,
    color_mode='grayscale',
    class_mode='categorical'
)
# Define the classifier model
classifier = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(
        filters=32,
        kernel_size=3,
        padding="same",
        activation="relu",
        input_shape=[128, 128, 1]
    ),
    tf.keras.layers.MaxPooling2D(pool_size=2, strides=2, padding='valid'),
    tf.keras.layers.Conv2D(filters=32, kernel_size=3, padding="same", activation="relu"),
    tf.keras.layers.MaxPooling2D(pool_size=2, strides=2, padding='valid'),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(units=128, activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(units=96, activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(units=64, activation='relu'),
    tf.keras.layers.Dense(units=27, activation='softmax')
])

# Compile the model
classifier.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
# Print model summary
classifier.summary()
# Train the model
SH=classifier.fit(
    training_set,
    epochs=5,
    validation_data=test_set
)
```

```
score=classifier.evaluate(test_set)
# Saving the Model
classifier.save('model.h5')
# Depict loss vs val_loss on line chart
import matplotlib.pyplot as plt
import pandas as pd
r1=pd.DataFrame(SH.history)
r1['Epochs']=SH.epoch
plt.plot(r1['Epochs'],r1['loss'],label='train')
plt.plot(r1['Epochs'],r1['val_loss'],label='test')
plt.xlabel("Epochs")
plt.ylabel("loss")
plt.legend()
plt.grid()
plt.show()
# Depict accuracy vs val_accuracy on line chart
plt.plot(r1['Epochs'],r1['accuracy'],label='train')
plt.plot(r1['Epochs'],r1['val_accuracy'],label='test')
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.grid()
plt.show()
```

Code for the System Design:

```
# Import necessary Libraries
import numpy as np
import cv2
import os
import PIL
from PIL import ImageTk
import PIL.Image
import speech_recognition as sr
import pyttsx3
from itertools import count
import string
from tkinter import *
import time
import pyttsx3
try:
    import tkinter as tk
except:
    import tkinter as tk
import tkinter as tk
import numpy as np
#Initialize the text-to-speech engine
text_speech= pyttsx3.init()
#Define image dimensions
image_x, image_y = 64,64
#load the pre-trained Keras model
from keras.models import load_mode
classifier = load_model('model.h5')
#Function to determine character similarity
def give_char():
    #check similarity of input with items in the file map
    import numpy as np
    from keras.preprocessing import image
    from keras.preprocessing.image import load_img
    test_image = load_img('tmp1.png', target_size=(64, 64))
    test_image = image.img_to_array(test_image)
    test_image = np.expand_dims(test_image, axis=0)
    result = classifier.predict(test_image)
    print(result)
    char_map ={i: chr(ord('A')+i) for i in range(26)}
    indx=np.argmax(result[0])
    return char_map.get(indx,' ')
def check_sim(i,file_map):
    for item in file_map:
        for word in file_map[item]:
            if(i==word):
                return 1,item
    return -1,""
#Define file paths for data and alphabet images
op_dest="D:\\PANDA\\filtered_data\\"
alpha_dest="D:\\PANDA\\alphabet\\"
# Class for the Start Page of the application
class StartPage(tk.Frame):
    def _init_(self, parent, controller):
        tk.Frame._init_(self,parent,bg="lightblue")
        label = tk.Label(self, text="Project of Artifical Intelligence For Deaf and
Aphonic",bg="lightblue" ,font=("Times New Roman", 32))
```

```
        label.pack(pady=10,padx=10)
          button = tk.Button(self, text="Deaf ",font=("Helvetica", 16),bg="lightyellow",
        command=lambda: controller.show_frame(VtoS))
        button.pack(pady=10)
            button2 = tk.Button(self, text="Apohnic ",font=("Helvetica",
          16),bg="lightyellow",command=lambda: controller.show_frame(StoV))
        button2.pack(pady=10)
        # parent('-fullscreen',True)
        button.config(width=15, height=1)
        button2.config(width=15, height=1)
        load = PIL.Image.open("bg1.png")
        load = load.resize((800,500))
        render = ImageTk.PhotoImage(load)
        img = Label(self, image=render,bg="lightblue")
        img.image = render
        img.place(x=400, y=270)
```

# Class for Voice to Sign functionality

```
class VtoS(tk.Frame):
    def _init_(self, parent, controller):
        cnt=0
        gif_frames=[]
        global inputtxt
        button2.pack(pady=10)
        def gif_stream():
            global cnt
            global gif_frames
            if(cnt==len(gif_frames)):
                return
            img = gif_frames[cnt]
            cnt+=1
            imgtk = ImageTk.PhotoImage(image=img)
            gif_box.imgtk = imgtk
            gif_box.configure(image=imgtk)
            gif_box.after(50, gif_stream)
        def hear_voice():
            store = sr.Recognizer()
            with sr.Microphone() as s:
                try:
                    print("listening ....")
                    audio_input = store.listen(s,timeout=5)
                    print("recording complete....")
                    text_output = store.recognize_google(audio_input, language='en-
                    US')
                    inputtxt.insert(END,text_output)
                except sr.WaitTimeoutError:
                    print("Timeout error. No speech detected")
                except sr.RequestError as e :
                    print(f"Could not request results from Google Web Speech
                  API;{e}")
                except sr.UnknownValueError:
                    print("Unable to recoginze speech")
                except Exception as e :
                    print(f"An error occured:{e}")
```

```
                inputtxt.insert(END, '')
```

# Class for Sign to Voice functionality

```
class StoV(tk.Frame):
    def _init_(self, parent, controller):


        # start_vid.pack()
        def start_video():
            video_frame = tk.Label(self)
            cam = cv2.VideoCapture(0)
            global img_counter
            img_counter = 0
            global img_text
            img_text = ''
            def video_stream():
                global img_text
                global img_counter
                if(img_counter>50):
                    cam.release()
                    return None
                img_counter+=1
                ret, frame = cam.read()
                frame = cv2.flip(frame,1)
                img=cv2.rectangle(frame,  (425,100),(625,300),  (0,255,0),  thickness=2,
                lineType=8, shift=0)
                lower_blue = np.array([35,10,0])
                upper_blue = np.array([160,230,255])
                imcrop = img[102:298, 427:623]
                hsv = cv2.cvtColor(imcrop, cv2.COLOR_BGR2HSV)
                mask = cv2.inRange(hsv, lower_blue, upper_blue)
                cv2.putText(frame, img_text, (30, 400),
                cv2.FONT_HERSHEY_TRIPLEX, 1.5, (0, 255, 0))
                img_name = "tmp1.png"
                save_img = cv2.resize(mask, (image_x, image_y))
                cv2.imwrite(img_name, save_img)
                time.sleep(0.1)
                tmp_text=img_text[0:]
                img_text = give_char()
                if(tmp_text!=img_text):
                    print(tmp_text)
                    disp_txt.insert(tk.END, tmp_text)
                img = PIL.Image.fromarray(frame)
                imgtk = ImageTk.PhotoImage(image=img)
                video_frame.imgtk = imgtk
                video_frame.configure(image=imgtk)
                video_frame.after(1, video_stream)
            video_stream()
            disp_txt.pack()
            video_frame.pack()
        start_vid = tk.Button(self,height = 2,width = 20, text="Start
Video",command=lambda: start_video())
        start_vid.pack()
```

# Start the Tkinter application

```
app = Tk_Manage()
app.geometry("800x750")
app.mainloop()
```

4.3 Testing

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses specific testing requirement.

4.3.1 Testing Methodology

Testing is a critical phase in the development and implementation of a blockchain-based e-voting system. The methodology employed must be comprehensive, covering various aspects such as unit testing, integration testing, security testing, performance testing, functional testing, usability testing, regression testing, auditability testing, disaster recovery testing, compliance testing, and interoperability testing.

Unit Testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Integration Testing Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications,

e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

4.3.2 Types of Testing

- **Unit testing**

Involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. Unit tests perform basic tests at component level and test a specific business process, application, and system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

- **Functional test**

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

• System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

• White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

• Black Box Testing

Black Box Testing is testing the software without any knowledge of the innerworkings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document.

CHAPTER 5

RESULT

In this report, a functional real time vision based American sign language recognition for Deaf and Aphonic people have been developed for asl alphabets. We achieved an accuracy of **95.8%** on our model.

5.1 Output Screen

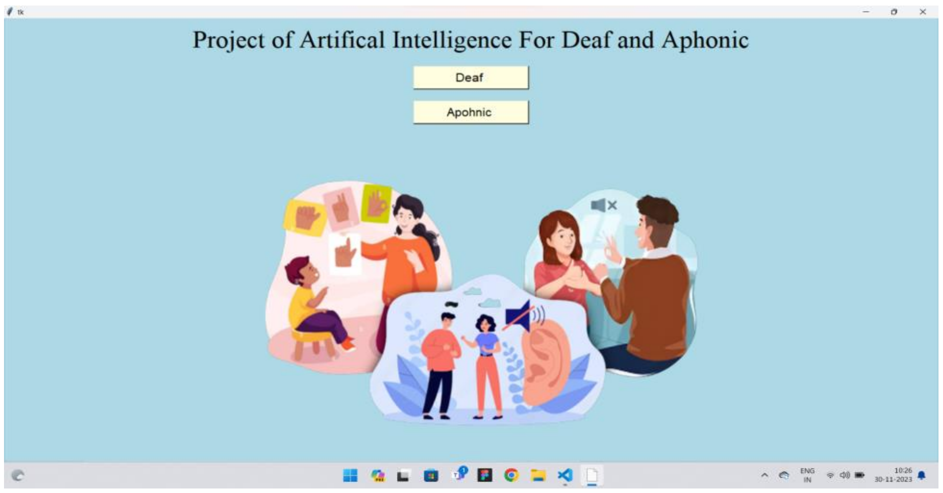

Figure – 5.1 Starting interface of the Application

The initial interface, marked as figure 5.1, serves as the launchpad for the project. This interface prompts users to select between two options: Upon selecting the "Deaf" option, the system transitions to the 5.2 figure, displaying a tailored screen catering specifically to the needs of the deaf community. Likely, choosing the "Aphonic" option prompts the system to present the 5.3 figure, showcasing a distinct screen designed to cater to the requirements of the aphonic user group.

The 5.2 figure is the screen for the deaf option. It is used to communicate with the deaf people. In this people can give the text and audio as the input. If we give the text as a input it split into words and if the word is the common and which we given in the dataset that gif will display or else finger spelling while display in the video format.

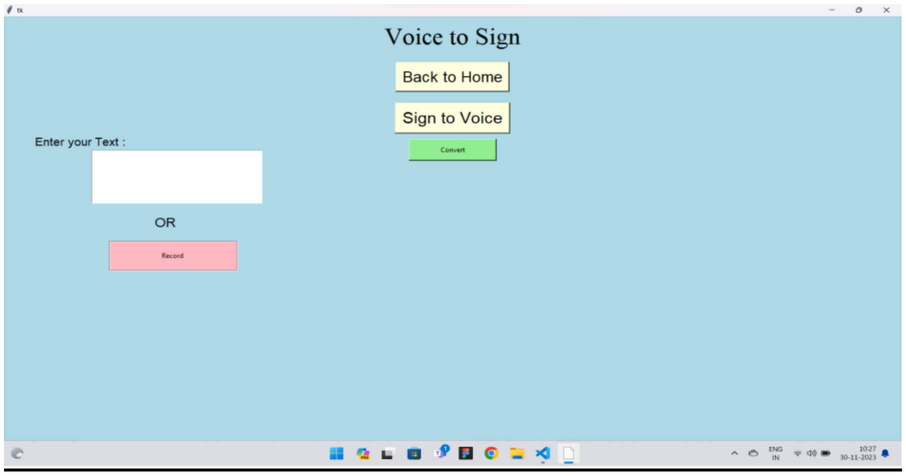

Figure – 5.2 The Screen for the Deaf People

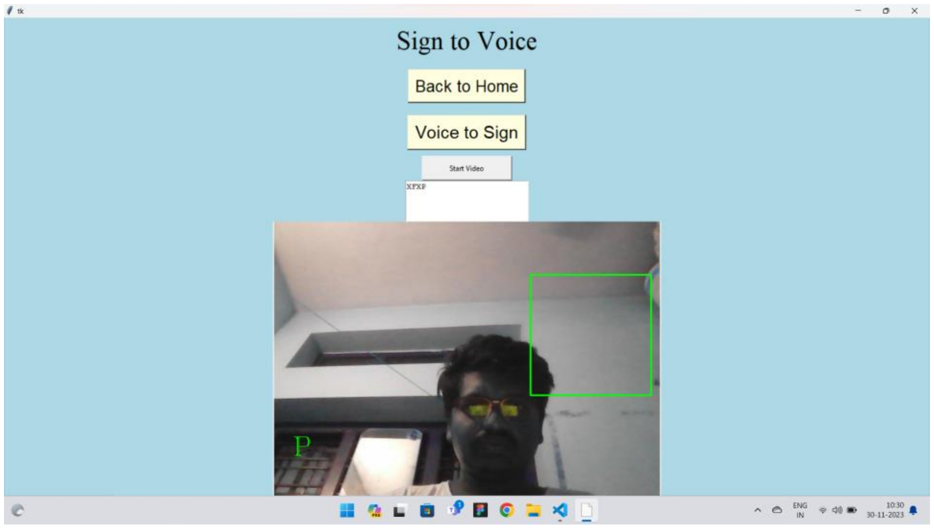

Figure – 5.3 The Screen for the Aphonic People

The above 5.3 figure is the screen for the aphonic people. In this the aphonic people can communicate with the normal it will help to convert the signs into text and audio. It is still in developing stage it will be now take just the letter and tell that letter with the accuracy of 50 % it will improve and the for this screen we need the good lighting condition and the good plain white background. Some deviation in the light also taking as symbol we will improve that in major project.

CONCLUSION

In this report, a functional real time vision based American Sign Language recognition for Deaf and Aphonic people have been developed for asl alphabets.

We achieved final accuracy of **98.0%** on our data set. We have improved our prediction after implementing the text or audio to the sign language, wherein we have verified and predicted symbols which are more similar to each other.

This gives us the ability to detect almost all the symbols provided that they are shown properly, there is no noise in the background and lighting is adequate.

FUTURE ENHANCEMENT

We are planning to achieve higher accuracy even in case of complex backgrounds by trying out various background subtraction algorithms.

We are also thinking of improving the Pre-Processing to predict gestures in low light conditions with a higher accuracy.

This project can be enhanced by being built as a web/mobile application for the users to conveniently access the project. Also, the existing project only works for ASL; it can be extended to work for other native sign languages with the right amount of data set and training. This project implements a finger spelling translator; however, sign languages are also spoken in a contextual basis where each gesture could represent an object, or verb. So, identifying this kind of a contextual signing would require a higher degree of processing and natural language processing (NLP).

In Major project we will try to develop the model which take sign as the input as give the text or audio as the output with proper word suggestions it helps the user for more better understanding and learning the sign language.

REFERENCES

- Article Artificial Intelligence Technologies for Sign Language Ilias Papastratis, Christos Chatzikonstantinous, Dimitrios Konstantinidis, Kosmas Dimitropoulos and Petros Daras sensors 2021.
- International Journal for Technological Research In Engineering Volume 7, Issue 12, August-2020
- Sign Language Recognition by Image Processing 1Sabari Priya A, 2Adrija Nair, 3Sreejin Madhavan, 4Kavitha Issac IJCRT | Volume 11, Issue 6 June 2023.
- E. Rajalakshmi et al.: Multi-Semantic Discriminative Feature Learning for Sign Gesture Recognition volume 11| 2023.
- N. Naz et al.: SIGNGRAPH: An Efficient and Accurate Pose-Based Graph Convolution Approach Toward SLR volume 11 | 2023.
- Number System Recognition (https://github.com/chasinginfinity/number-sign-recognition)
- https://opencv.org/
  https://en.wikipedia.org/wiki/TensorFlow
- https://en.wikipedia.org/wiki/Convolutional_neural_nework
- https://giphy.com/search/sign-language