

DESIGNING AN REHABILITATION AID TO PROVIDE ASSISTANCE TO SPEECH AND AUDITORY DISABILITIES

A PROJECT REPORT

Submitted by

Sr. No.	Name	Enrollment No.
1.	Harshil Rajodiya	170130103092
2.	Chiranjit Rathod	170130103093
3.	Mitesh Vaghasia	170130103115

In fulfilment for the award of the degree

Of

BACHELOR OF ENGINEERING

IN

Bio-Medical Engineering



Government Engineering College, Gandhinagar

Gujarat Technological University, Ahmedabad



CERTIFICATE

Date:

This is to certify that the dissertation entitled “DESIGNING AN REHABILITATION AID TO PROVIDE ASSISTANCE TO SPEECH AND AUDITORY DISABILITIES” has been carried out by Harshil Rajodiya, Chiranjit Rathod, Mitesh Vaghasia under my guidance in fulfilment of the degree of Bachelor of Engineering in 8th semester of Gujarat Technological University, Ahmedabad during the academic year 2020-2021.

Guide:

Prof. Manan M. Nanavati

Head of the Department

Prof. Ghanshyam D. Parmar

ACKNOWLEDGEMENT

We are very grateful to our guide Prof. Manan M. Nanavati who has endeavoured us from the learning and processing the knowledge during the course of project work. Without their support and also their invaluable guidance, comments, stimulating suggestions and encouragement throughout the course of the project, we would not have been reaching this satisfactory result.

We would also like to thank Prof. Ghanshyam D Parmar, Head of Department, Bio Medical Engineering, Government Engineering College, Gandhinagar, who gave opportunity to do this project at an extreme organization.

We would also like to acknowledge with much appreciation the crucial role of the other staff members of the institute who rendered their help during the period of project work. Last but not the least, we wish to avail this opportunity to appreciate and give a special thanks to our team members for their supportive contribution, project enhancing comments and tips which had improved the presentation skills, report writing and also brought clarity in the software work.

ABSTRACT

The sign language is one of the mostly accepted method of communication for people with speech and hearing disability. Sign language has received an incredible attention and advancement which has grown over the years. On the other side if we opt for the human translation service it can cost a money (may be proved as expensive). To solve this, we need such kind of product that is both versatile and robust. This project proposes a method with an objective of eliminating the barrier between the deaf and dumb and the rest by making a sign language recognition system or a translator. This project shows the recognizing various hand gestures in Indian Sign Language using deep learning-based architecture i.e., CNN. The most challenging part in the designing of a sign language translator is of designing a good classifier that can classify the input static gestures with high accuracy and of minimum possible error. We present an approach towards sign language recognition which uses Convolutional Neural Network (CNN), which is trained on data collected using webcam. The system trained CNNs for the classification of numbers, alphabets and other daily used words using 17113 images. The most challenging part to sign language translator is of designing a good classifier that can classify the input static gestures with high accuracy and of minimum possible error. The proposed system has trained the classifier with different parameters and had attained a good accuracy. In our method, the hand is first passed through a filter and after the filter is applied the hand is passed through a classifier which predicts the class of the hand gestures. Our method provides 96% accuracy for the 27 letters of the alphabet. The result also shows that with increasing the number of images (i.e., it can be pre-processed images also) in dataset, results into increase in the accuracy of the system.

INDEX

Sr. No.	Content	Page No.
1.	Introduction	1
2.	Aim, Objective, Motivation	3
3.	Literature Survey	4
4.	Methodology	7
5.	Project Description	11
6.	Result	19
7.	Conclusion	21
8.	Issues and challenges	22
9.	Future work	22
10.	References	23
11.	Appendix	24

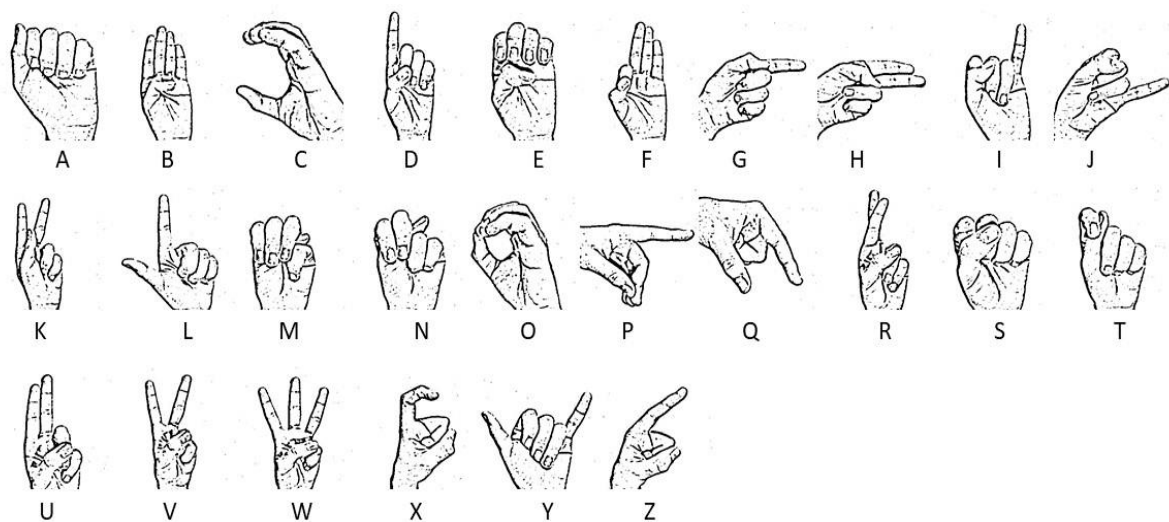
INTRODUCTION

In our daily life, the communication between several different communities is fundamental and very much important to share information. Being able to communicate effectively is a vital life skill but for the people with speech and hearing disability, they find it difficult to convey their messages with others. The process of communication between two people can be done using various medium. Not everyone knows how to interpret a sign language when having a conversation with such community like of deaf and dumb person. One finds it difficult to communicate without an interpreter or some other sources. We need to convert the sign language so that it is understood by others and also help them to communicate without any barriers. One of the effective solutions of this difficulty is sign language recognition system. Using sign language different gestures of hand are used to express meaningful information. Language of sign is different in different parts of world. There are 135 sign languages prevalent throughout the world for communication. Each sign language is different from the other like American Sign Language used in America is different from the Indian Sign Language of India. Looking to the ease of understanding Indian Sign Language, we standardized to work on Indian Sign Language gestures. We need to convert the sign language so that it is understood by others and also help them to communicate without any barriers.

Sign language recognition is still a challenging problem inspire of many research efforts during the last many years. One of the methods of hand gesture recognition is to use the hand gloves for human computer interaction. But this method is sophisticated as it requires user to wear glove and carry a load of cables connecting the device to a computer. Therefore, to eliminate this complication and to make user interaction with computer easy and natural we proposed to work on sign recognition using bare hands i.e., no usage of any external wearable hardware. Mainly sign language recognition processes are highly depending on human based translation services. The involvement of human expertise is very difficult and expensive also for translation. Now our proposed automatic sign language recognition system leads to understand the meaning of different signs without any aid from the expert. In common, any sign language recognition system contains several modules like object tracking, skin segmentation, feature extraction, and recognition. The first two modules

are basically used to extract and locate hands in the video frames and the next modules is used for feature extraction, classification and recognition of gesture. For an image-based gesture recognition system, image space variables are widely large, it is crucial to extract the essential features of the image.

In our project we basically focus on producing a model which can recognise Finger spelling-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.



AIM

To create a Sign Language Translator by using Convolution Neural Network which can be used with embedded devices having less resources.

OBJECTIVE

The main objective of our project is to contribute to the field of automatic sign language recognition. Our focus is mainly on the recognition of the real time sign language gestures. This work focused on deep learning approach to numbers, alphabets and most often used words. through this work we want to ease the interaction for people with speech and hearing disabilities and also other objective is to convert sign language into text.

MOTIVATION

The motivation behind building such a system includes:

- Sign-to-text translation system or dialog systems which can be used in various public domains like in Government Departments, airports etc.
- Such system can help to translate the video to text or even to speech that enables the inter communication between normal and deaf people.

LITERATURE SURVEY

In the recent years there has been tremendous research done on the hand gesture recognition. With the help of literature survey, we realized the basic steps in hand gesture recognition are: -

- Data acquisition
- Data pre-processing
- Feature extraction
- Gesture classification

Data acquisition:

The different approaches to acquire data about the hand gesture can be done in the following ways:

1. Use of sensory devices

- It uses electromechanical devices to provide exact hand configuration, and position. Different glove-based approaches can be used to extract information. But it is expensive and not user friendly.

2. Vision based approach

- In vision-based methods computer camera is the input device for observing the information of hands or fingers. The Vision Based methods require only a camera, thus realizing a natural interaction between humans and computers without the use of any extra devices. These systems tend to complement biological vision by describing 7 artificial vision systems that are implemented in software and/or hardware.
- The main challenge of vision-based hand detection is to cope with the large variability of human hand's appearance due to a huge number of hand movements, to different skin-colour possibilities as well as to the variations in viewpoints, scales, and speed of them camera capturing the scene.

Data pre-processing and Feature extraction for vision-based approach:

In the approach for hand detection combines threshold-based color detection with background subtraction. We can use Ad boost face detector to differentiate between faces and hands as both involve similar skin-color. We can also extract necessary image which is to be trained by applying a filter called Gaussian blur. The filter can be easily applied using open computer vision also known as OpenCV and is described in.

For extracting necessary image which is to be trained we can use instrumented gloves as mentioned in. This helps reduce computation time for pre-processing and can give us more concise and accurate data compared to applying filters on data received from video extraction.

We tried doing the hand segmentation of an image using color segmentation techniques but as mentioned in the research paper skin color and tone is highly dependent on the lighting conditions due to which output we got for the segmentation we tried to do were no so great. Moreover, we have a huge number of symbols to be trained for our project many of which look similar to each other like the gesture for symbol 'V' and digit '2', hence we decided that in order to produce better accuracies for our large number of symbols, rather than segmenting the hand out of a random background we keep background of hand a stable single color so that we don't need to segment it on the basis of skin color. This would help us to get better results.

Gesture classification:

In Hidden Markov Models (HMM) is used for the classification of the gestures. This model deals with dynamic aspects of gestures. Gestures are extracted from a sequence of video images by tracking the skin-colour blobs corresponding to the hand into a body – face space centred on the face of the user. The goal is to recognize two classes of gestures: deictic and symbolic. The image is filtered using a fast look-up indexing table. After filtering, skin colour pixels are gathered into blobs. Blobs are statistical objects based on the location (x, y) and the colourimetry (Y, U, V) of the skin colour pixels in order to determine homogeneous areas.

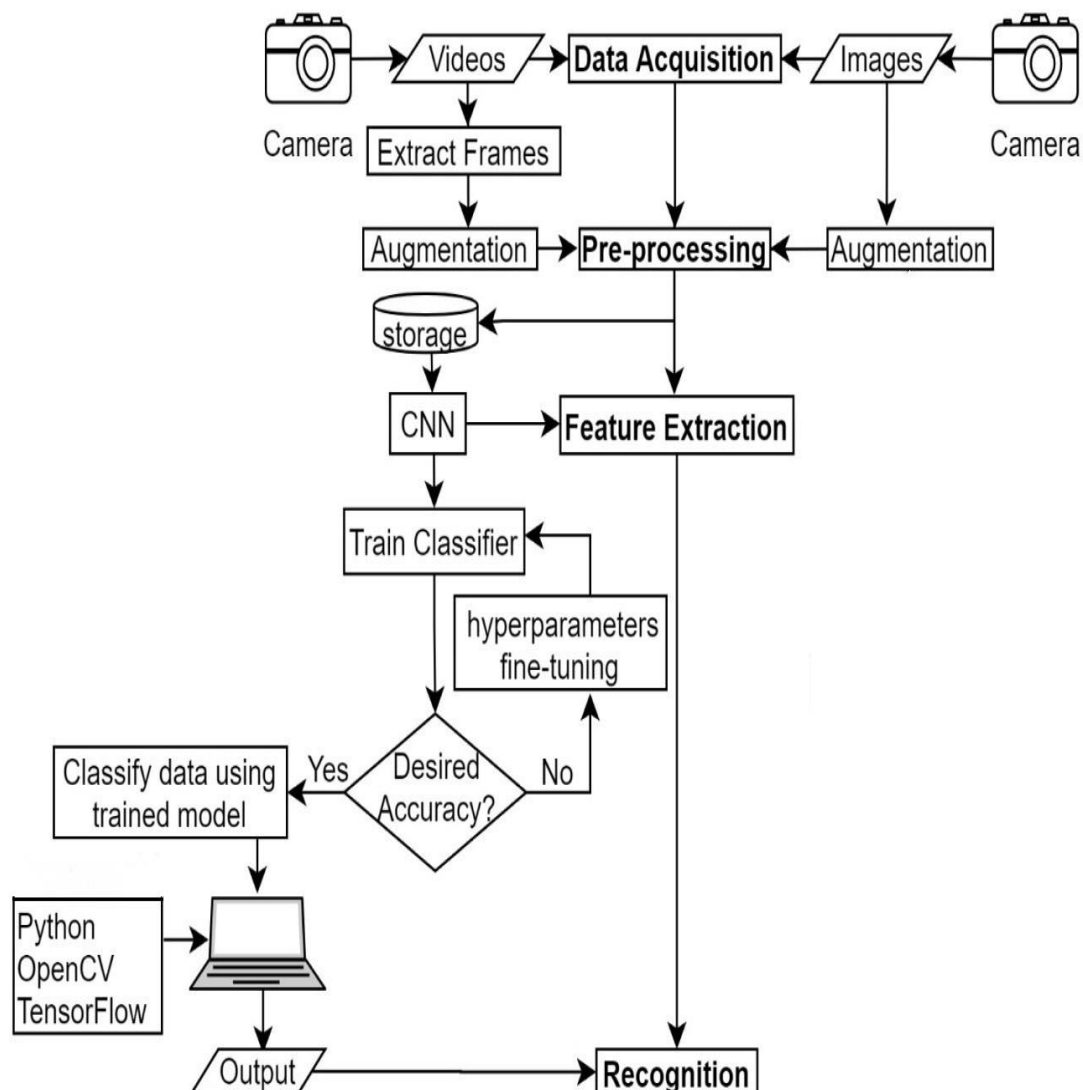
In Naïve Bayes Classifier is used which is an effective and fast method for static hand gesture recognition. It is based on classifying the different gestures according to geometric based invariants which are obtained from image data after segmentation. Thus, unlike many other recognition methods, this method is not dependent on skin colour. The gestures are extracted from each frame of the video, with a static background. The first step is to segment and label the objects of interest and to extract geometric invariants from them. Next step is the classification of gestures by using a K nearest neighbour algorithm aided with distance weighting algorithm (KNNDW) to provide suitable data for a locally weighted Naïve Bayes" classifier.

According to paper on "Human Hand Gesture Recognition Using a Convolution Neural Network" by Hsien-I Lin, Ming-Hsiang Hsu, and Wei-Kai Chen graduates of Institute of Automation Technology National Taipei University of Technology Taipei, Taiwan, they construct a skin model to extract the hand out of an image and then apply binary threshold to the whole image. After obtaining the threshold image they calibrate it about the principal axis in order to centre the image about it. They input this image to a convolutional neural network model in order to train and predict the outputs. They have trained their model over 7 hand gestures and using their model they produce an accuracy of around 95% for those 7 gestures.

METHODOLOGY

Our work is based on the vision-based approach. Image processing-based algorithms are effectively used to recognise the hand gestures of the signer. This technique is simpler to implement as there is no need to wear any extra hardware. However, on the other side there are chances of accuracy problems related to the image processing algorithms.

We have used appearance-based approach in vision-based sign language recognising system. It makes the usage of images as an input. They directly elucidate from these images or videos. This approach doesn't require any spatial representation of the body likewise in the 3-D based approach where it used 3-D information of key elements of the body parts.



It is divided into four different stages which includes data acquisition, pre-processing, feature extraction, and recognition. The images/video frames are provided as input and corresponding output is being detected and respective text of the recognize sign/gesture is displayed. Images used to train the model was augmented before the actual training part. Augmentation adds the variations in the dataset so the CNN model can be trained in more effective way. Then the trained model is saved and particularly loaded with OpenCV to recognize the digits in real time. Let's take a brief look at the four stages mentioned above.

1. Data Acquisition – We have tried to obtained our dataset but due to the lake of resources we opted for performing our pre-processing method directly on to the existing dataset.

2. Pre-processing - while training the model requirement of data is very large in order to work in very effective manner. So, if we have a limited number of images in our dataset for our network, therefore in order to increase the data set we have generally augmented our images. We have just made minor alterations to our dataset like flips, shifts or rotations. Data augmentation can also help in reducing the chances of overfitting on models. Here we have resized and rescale our images to treat all images in same manner.

3. Feature Extraction - features from the images are extracted using CNN algorithm.

4. Recognition - for this purpose trained model was loaded on a laptop using TensorFlow as a backend and with the help of OpenCV frames of real time hand shaped video are captured. Hence model detects and predicts the inputted hand gestures correctly.

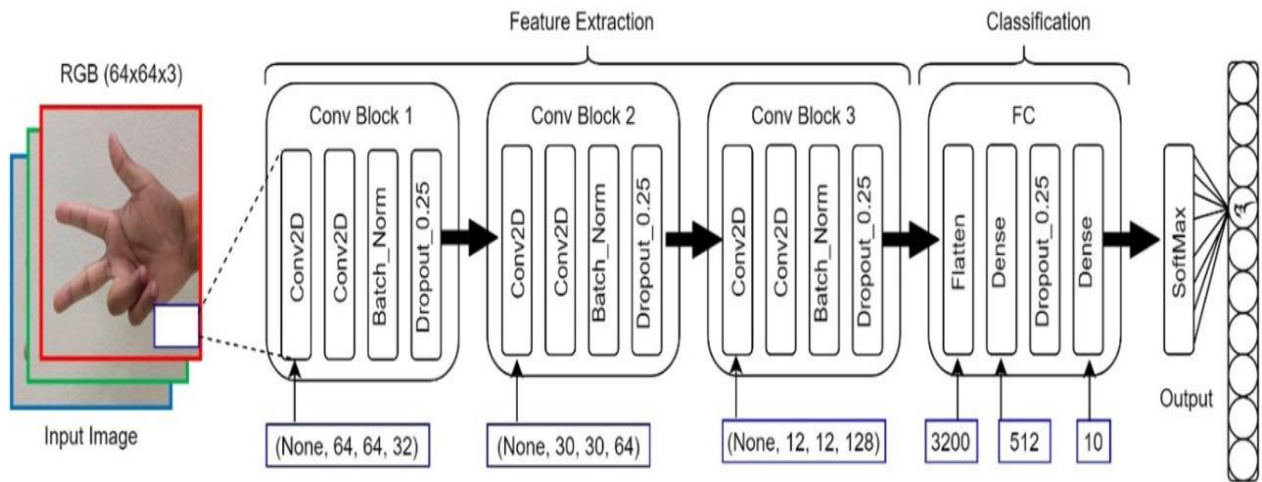
Hardware/Software Logic

Machine Learning cannot be extensively used for feature extraction because its algorithm cannot handle high dimensional data and also it is one of the big challenges in object detection, image classification, etc.

Deep Learning inspired from how biological neural network have been playing a key role in the field of machine learning. As we are aware of the fact that images are high dimensional vectors. It would take a huge number of parameters to characterize the network, if we take flatten image feature vector 1-D. To address this problem (of taking long vector), CNN are proposed to reduce the number of parameters and adapting the network architecture specifically to vision tasks. CNN works well on tasks.

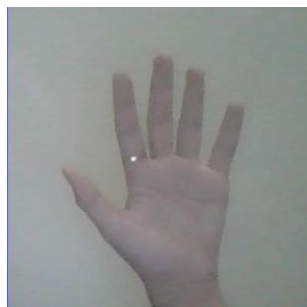
Convolution Neural Network basically includes set of layers each having their own functionalities. CNN works by extracting features from images. There is no need for manual feature extraction. Features are trained while the network trains on the set of images. CNNs key components includes:

- Convolutional layer - consist of grouped neuron in a rectangular grid. It is the application of filter to an input that results in an activation. Repeated activation of it results into a feature map, which indicating the locations and strength of a detected features in an input that is image.
- Activation function - decides whether a neuron should fire an output or not and it is also used to increase the non-linearity into the output of the neuron.
- Pooling layers - are present after a single or a set of convolution layers. Its function is to down sample the detection of features in feature map.
- Fully connected layer– dense layer which is the final learning phase where classification takes place. Every node of it is connected to every nodes of previous layer.

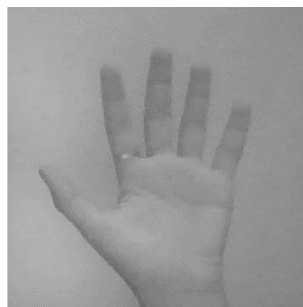


Dataset Generation:

For our project we tried to make our own dataset for the ISL language but due to lack of resources we were unable to do so. Then we find out the already existing datasets that matched our requirements. All we could find were the datasets in the form of RGB values. Hence, we decided to transform it into our required form. By using batch mode transformation/data augmentation we try to convert it into Gray scale Image as shown below:



Raw Image
(RGB)



Gray Scale Image



Gaussian Blur

PROJECT DESCRIPTION

Implementation of Model:

Importing libraries:

To use the power of libraries and packages, firstly we have imported all the required libraries to have an ease in performing our task.



```
# 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
```

CNN architecture summary:

It is one of the most important part for our network i.e. in defining our required architecture. A brief explanation of CNN is provided below along with a figure of our model architecture summary:

- * starts with an input image.
- * applies many different filters to obtain a feature map.
- * applies a RELU function to increase non-linearity.
- * applies pooling layer to each future map.
- * flattening the pooled images into one long vector.
- * inputs the vector into fully connected ANN.
- * dropout is also used to mitigate the overfitting.
- * then final fully connected layer provides the voting of the classes.
- * trains through forward propagation and backpropagation for many epochs.

Model: "sequential"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	320
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 32)	9248
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 32)	0
flatten (Flatten)	(None, 28800)	0
dense (Dense)	(None, 128)	3686528
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 96)	12384
dropout_1 (Dropout)	(None, 96)	0
dense_2 (Dense)	(None, 64)	6208
dense_3 (Dense)	(None, 27)	1755
Total params: 3,716,443		
Trainable params: 3,716,443		
Non-trainable params: 0		

Training our network:

After successfully completing all the steps of constructing our network. Now the next step is to train our network. To train the model effectively we have set epochs and other necessary required configurations to train it effectively.

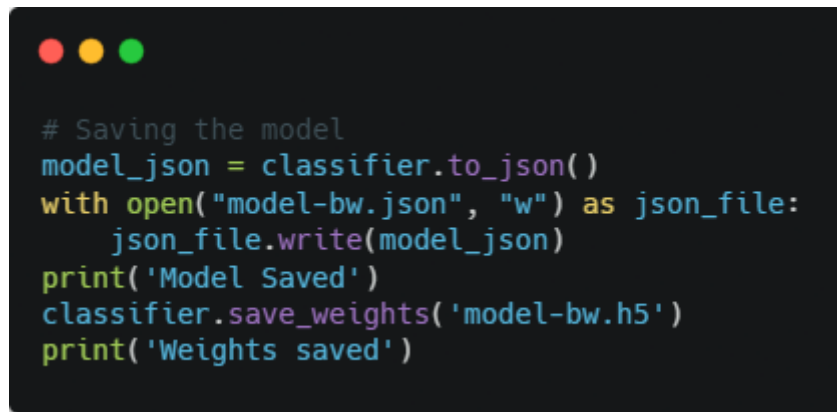
```

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 model:

The metadata file (or model.json) in a Common Data Model folder describes the data in the folder, metadata and location, as well as how the file was generated and by which data producer. Metadata summarizes basic information about data, which can make finding and working with particular instances of data easier. JSON (JavaScript Object Notation) is a popular data format used for representing structured data. So, keeping in the mind regarding usability of data model we have saved our model as json file.

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It contains Python code for saving a model.

```
# 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')
```

Gesture Classification:

Our approach uses two layers of algorithm to predict the final symbol of the user.

Algorithm Layer 1:

- 1) Apply gaussian blur filter and threshold to the frame taken with OpenCV to get the processed image after feature extraction.
- 2) 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.
- 3) Space between the words is considered using the blank symbol.

Algorithm Layer 2:

- 1) We detect various sets of symbols which show similar results on getting detected.
- 2) We then classify between those sets using classifiers made for those sets only.

Layer 1:

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 $30 \times 30 \times 32 = 28800$ 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 Layer:

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 overfitting.

Dropout Layer:

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

Layer 2:

We are using two layers of algorithms to verify and predict symbols which are more similar to each other so that we can get us close as we can get to detect the symbol shown. 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

So, to handle above cases, we made three different classifiers for classifying these sets:

1. {D, R, U}
2. {T, K, D, I}
3. {S, M, N}

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

Autocorrective Feature:

A python library Hunspell_suggest is used to suggest correct alternatives for each (incorrect) input word and we display a set of words matching the current word in which the user can select a word to append it to the current sentence. This helps in reducing mistakes committed in spellings and assists in predicting complex words.

Training and Testing:

We convert our input images (RGB) into grayscale and apply gaussian blur to remove unnecessary noise. Then we have applied 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 values in each class equals 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 i.e., Adam Optimizer.

User Interface;

UI stands for user interface and is the point of communication between human and the computer. The function of our UI is to provide users with an intuitive interaction and support that manages to provide a solution for the desired task.

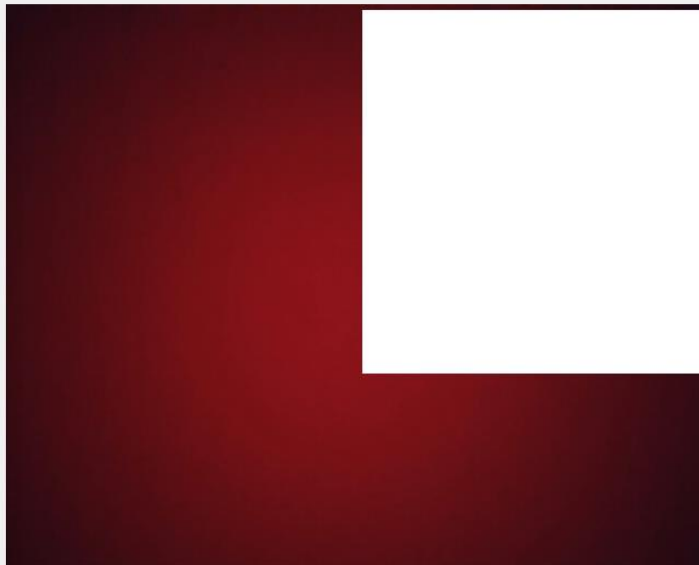
We have provided user input control's which are the interactive component of interface. It basically includes space for input gesture, various other spaces for character, word, sentence and buttons for audio output, backspace, reset and also for the suggestions to be picked up from the screen.

We are providing both text and audio output on the same platform which makes a user to understand the input gestures more effectively.

As you can see the UI window below. At the top there is a title shown and exactly below it there is a space provided for gestures to displayed. And the on the right-hand side of it three buttons are given with the distinctive functionality which are as follow i.e., pressing the audio button one can have an audio output; backspace button to correct the wrong word being predicted; reset button to have a blank screen for new gestures to be classified.

After that downside in the UI screen there are spaces given for result to be obtained i.e., for character, word, sentence. And below it few more spaces are provided for the suggestions to presented before a user to help me picking up the correct word in order to communicate effectively.

Sign Language to Text



About

Audio

Backspace

Reset

Character: blank

Word:

Sentence:

Suggestions



RESULT

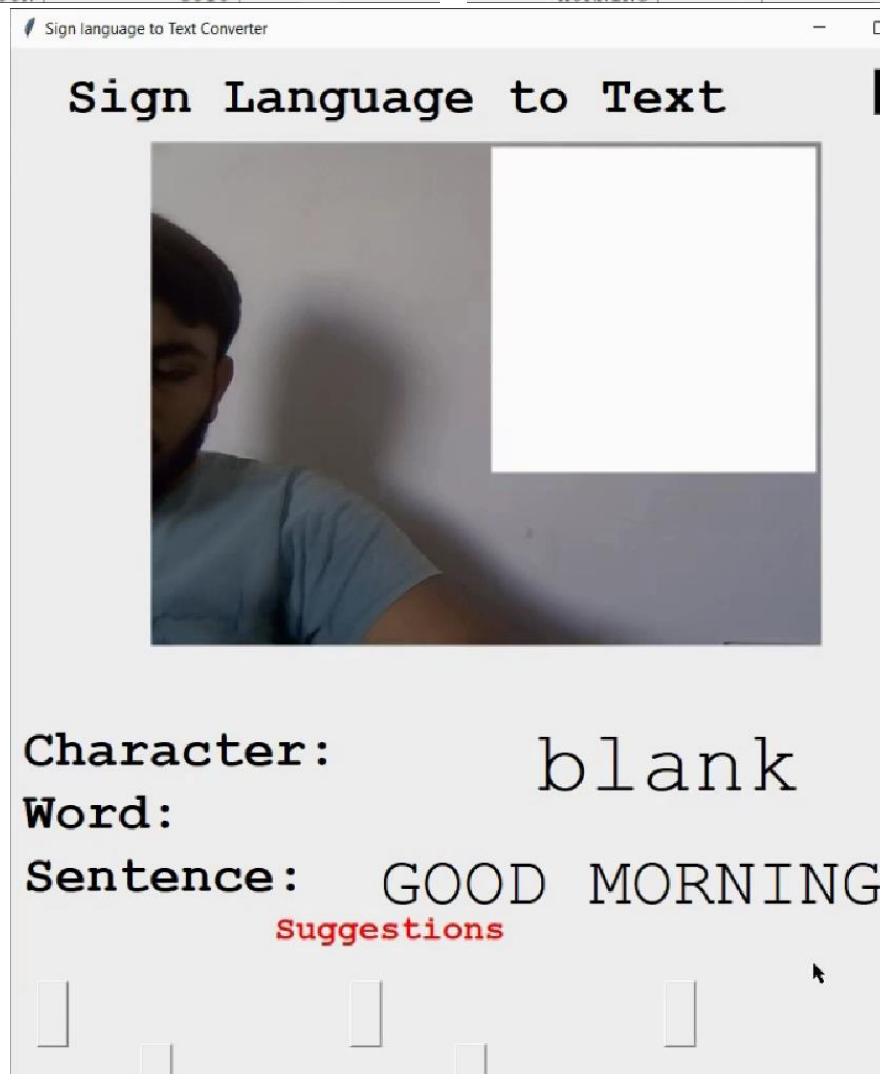
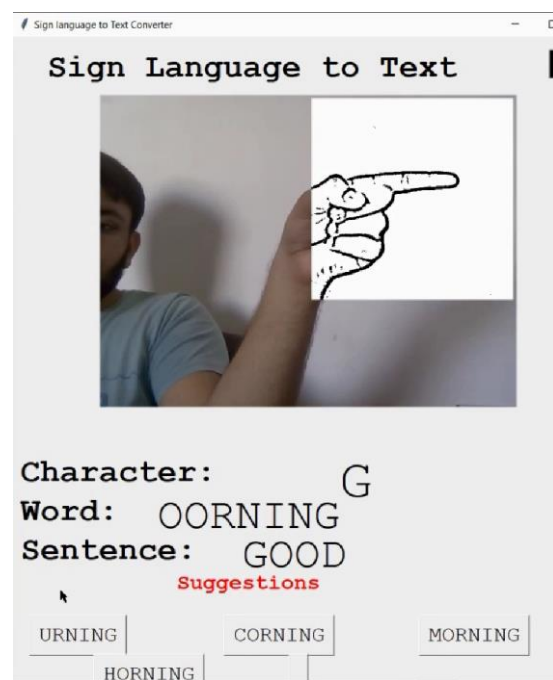
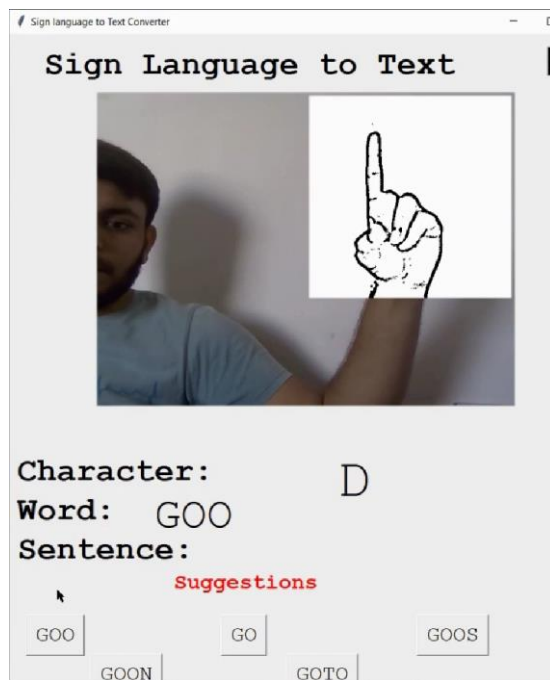
We have achieved an accuracy of 95.8% in our model using only layer 1 of our algorithm and by using the combination of layer 1 and layer 2 we achieve an accuracy of 98.0%.

We have trained the model for different gestures with set of two algorithm layers. Our experimental evaluation was performed for all the gestures and we also try to form sentence which shown below.

As we can see on the window screen displaying a blank output in the character space. This is because user is doing nothing i.e., performing no gesture as an input to the system.

After that when a user start providing input, simultaneously we can also see the suggestions being displayed below from where a user can select the output word if the provided input sign does not give the appropriate output text.

So, after that we have try to write a whole sentence comprising of two words i.e., GOOD MORNING. This is shown below stepwise in the figures. And along with it we have also produced audio output for the same. This step can be done to form other words also.



CONCLUSION

Communication between deaf-mute and a normal person have always been a challenging task. The goal of our project is to reduce the barrier between them. We have made our effort by contributing to the field of Indian Sign Language recognition. In this project, we developed a CNN-based human hand gesture recognition system. The salient feature of our system is that there is no need to build a model for every gesture using hand features such as fingertips and contours. Here in this project, we have constructed a CNN classifier which is capable of recognizing sign language gestures. The proposed system has shown satisfactory results on the transitive gestures.

In this report, a functional real time vision-based sign language recognition for deaf and dumb people have been developed. We achieved final accuracy of 98.0% on our dataset. We are able to improve our prediction after implementing two layers of algorithms, we have also verified our result for the similar looking gesture which were more prone to misclassification. This way we are able to detect almost all the symbols provided that they are shown properly, there is no noise in the background and lighting is adequate.

ISSUES AND CHALLENGES

There were many challenges faced by us during the project. The very first issue we faced was of dataset. So as having the most suitable dataset can get you through the good accuracy in predicting the gestures. Second issue was to select a filter which we could apply on our images so that proper features of the images could be obtained and hence then we could provide the images as input for CNN model. We tried various filter including binary threshold, canny edge detection, gaussian blur, etc; but finally, we settled with gaussian blur filter. More issues were faced relating to the accuracy of the model we trained in earlier phases which we eventually improved by increasing the input image size and also by improving the dataset. And further there was also issues regarding transitive gesture recognition in which the gesture we try to opt for the prediction was not getting stabilize in the frame but was soon tackle by adjusting the frame time and threshold value.

FUTUREWORK

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.

REFERENCE

- [1] Aman Aryan and Subham Sanghai, “Indian Sign Language Recognition by Feature Extraction Using SURF”, Birla Institute of Technology, Mesra.
- [2] Beena M.V. and Dr. M.N. Agnisarman Namboodiri, “Automatic Sign Language Finger Spelling Using Convolution Neural Network: Analysis”, 2017 International Journal of Pure and Applied Mathematics.
- [3] Hsien-I Liny, Ming-Hsiang Hsu, and Wei-Kai Chen, “Human Hand Gesture Recognition Using a Convolution Neural Network”, 2014 IEEE International Conference on Automation Science and Engineering (CASE).
- [4] Brandon Garcia, Sigberto Alarcon Viesca, “Real-time American Sign Language Recognition with Convolutional Neural Networks”, 2016 Stanford University Stanford, CA.
- [5] Sirshendu Hore, Sankhadeep Chatterjee, V. Santhi and Nilanjan Dey, “Indian Sign Language Recognition using Optimized Neural Networks”, 2015 International Conference on Information Technology and Intelligent Transportation Systems.
- [6] Mahesh Kumar N B, “Conversion of Sign Language into Text”, International Journal of Applied Engineering Research ISSN 0973 -4562 Volume 13, Number 9 (2018) pp. 7154-7161.
- [7] Kuntal Kumar Pal, Sudeep K. S., “Pre-processing for Image Classification by Convolutional Neural Networks”, IEEE International Conference On Recent Trends In Electronics Information Communication Technology, May 20-21, 2016, India.

APPENDIX

```
import os
import cv2
import operator
import tkinter as tk
from PIL import Image, ImageTk
from keras.models import model_from_json
from hunspell import Hunspell
from string import ascii_uppercase
from gtts import gTTS
from playsound import playsound

class Application:
    def __init__(self):
        self.directory = "model/"
        self.hs = Hunspell('en_US')
        self.vs = cv2.VideoCapture(0)
        self.current_image = None
        self.current_image2 = None

        self.json_file = open(self.directory+"model.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(self.directory+"model.h5")

        self.json_file_dru = open(self.directory+"model_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(self.directory+"model_dru.h5")

        self.json_file_tkdi = open(self.directory+"model_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(self.directory+"model_tkdi.h5")

        self.json_file_smn = open(self.directory+"model_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(self.directory+"model_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 to Text Converter")
        self.root.protocol('WM_DELETE_WINDOW', self.destructor)
        self.root.geometry("1100x1100")

        self.canvas = tk.Canvas(width = 1100,height = 1100)
        self.canvas.pack(fill = "both", expand = True)

        self.panel = tk.Label(self.root)
        self.panel.place(x = 135, y = 90, width = 640, height = 480)
        self.panel2 = tk.Label(self.root) # initialize image panel
        self.panel2.place(x = 460, y = 95, width = 310, height = 310)

        self.canvas.create_text(450, 50, text = "Sign Language to Text",fill = "black",font=("courier",30,"bold"))

        self.panel3 = tk.Label(self.root) # Current Symbol
        self.panel3.place(x = 500,y=600)

        self.canvas.create_text(155, 653, text = "Character:",fill = "black",font=("courier",30,"bold"))

        self.panel4 = tk.Label(self.root) # Word
        self.panel4.place(x = 220,y=680)

        self.canvas.create_text(110, 713, text = "Word:",fill = "black",font=("courier",30,"bold"))
```

```

self.panel5 = tk.Label(self.root) # Sentence
self.panel5.place(x = 350,y=740)

self.canvas.create_text(140, 773, text = "Sentence:",fill = "black",font=("courier",30,"bold"))

self.T4 = tk.Label(self.root)
self.T4.place(x = 270,y = 800)
self.T4.config(text = "Suggestions",fg="red",font = ("Courier",20,"bold"))

self.btcall = tk.Button(self.root,command = self.action_call,height = 0,width = 0)
self.btcall.config(text = "About",bg="black",fg="white",font = ("Courier",14))
self.btcall.place(x = 950, y = 20)

self.bt1=tk.Button(self.root, bg= "#DAF7A6", activebackground='white',command=self.action1,height = 0,width
= 0)
self.bt1.place(x = 25,y=890)

self.bt2=tk.Button(self.root, bg= "#DAF7A6", activebackground='white',command=self.action2,height = 0,width
= 0)
self.bt2.place(x = 325,y=890)

self.bt3=tk.Button(self.root, bg= "#DAF7A6", activebackground='white',command=self.action3,height = 0,width
= 0)
self.bt3.place(x = 625,y=890)

self.bt4=tk.Button(self.root, bg= "#DAF7A6", activebackground='white',command=self.action4,height = 0,width
= 0)
self.bt4.place(x = 25,y=950)

self.bt5=tk.Button(self.root, bg= "#DAF7A6", activebackground='white',command=self.action5,height = 0,width
= 0)
self.bt5.place(x = 325,y=950)

self.bt6=tk.Button(self.root, text="Audio", bg= "#DAF7A6", activebackground='white', font = ("Courier",20))
self.bt6.place(x = 930,y=80)

self.bt7=tk.Button(self.root, text="Backspace", bg= "#DAF7A6", activebackground='white', font =
("Courier",20))
self.bt7.place(x = 880,y=140)

self.bt8=tk.Button(self.root, text="Reset", bg= "#DAF7A6", activebackground='white', font = ("Courier",20))
self.bt8.place(x = 930,y=200)

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",35))
        self.panel4.config(text=self.word,font=("Courier",25))
        self.panel5.config(text=self.str,font=("Courier",25))
        predicts=self.hs.suggest(self.word)

```

```

        predicts = self.loaded_model.predict(test_image)
        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.bt5.config(text=predicts[4],font = ("Courier",20))
        else:
            self.bt5.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] > 15):
    for i in ascii_uppercase:
        if i == self.current_symbol:
            print(i)
            continue
        tmp = self.ct[self.current_symbol] - self.ct[i]
        if tmp < 0:
            tmp *= -1
        if tmp <= 5:
            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 = ""
            print(self.str)
        def Text_to_speech():
            if os.path.exists("audio.mp3"):
                os.remove("audio.mp3")
            Message = self.str
            speech = gTTS(text = Message)
            speech.save('audio.mp3')
            playsound('audio.mp3')
        def erase():
            self.str = ""
        def Back_Space():
            self.str = self.str.rstrip(self.str[-1])
        self.bt6.config(command = Text_to_speech)
        self.bt7.config(command = Back_Space)
        self.bt8.config(command = erase)
    else:
        if(len(self.str) > 16):
            self.str = ""
            self.blank_flag = 0
            self.word += self.current_symbol
            print(self.str)

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.root1.destroy()

```



```

def action_call(self) :

    self.root1 = tk.Toplevel(self.root)
    self.root1.title("About")
    self.root1.protocol('WM_DELETE_WINDOW', self.destructor1)
    self.root1.geometry("900x900")

    self.tx = tk.Label(self.root1)
    self.tx.place(x = 360,y = 40)
    self.tx.config(text = "Efforts By", font = ("Courier",20,"bold"))

    self.photo1 = tk.PhotoImage(file='Pictures/chiranjit.png')
    self.w1 = tk.Label(self.root1, image = self.photo1)
    self.w1.place(x = 170, y = 105)
    self.tx6 = tk.Label(self.root1)
    self.tx6.place(x = 170,y = 310)
    self.tx6.config(text = "Chiranjit\n170130103093", font = ("Courier",15,"bold"))

    self.photo2 = tk.PhotoImage(file='Pictures/mitesh.png')
    self.w2 = tk.Label(self.root1, image = self.photo2)
    self.w2.place(x = 380, y = 105)
    self.tx2 = tk.Label(self.root1)
    self.tx2.place(x = 380,y = 310)
    self.tx2.config(text = "Mitesh\n170130103115", font = ("Courier",15,"bold"))

    self.photo3 = tk.PhotoImage(file='Pictures/harshil.png')
    self.w3 = tk.Label(self.root1, image = self.photo3)
    self.w3.place(x = 590, y = 105)
    self.tx3 = tk.Label(self.root1)
    self.tx3.place(x = 590,y = 310)
    self.tx3.config(text = "Harshil\n170130103092", font = ("Courier",15,"bold"))

    self.tx7 = tk.Label(self.root1)
    self.tx7.place(x = 220,y = 380)
    self.tx7.config(text = "Under the supervision of", font = ("Courier",20,"bold"))

    self.photo6 = tk.PhotoImage(file='Pictures/sir.png')
    self.w6 = tk.Label(self.root1, image = self.photo6)
    self.w6.place(x = 380, y = 430)
    self.tx6 = tk.Label(self.root1)
    self.tx6.place(x = 230,y = 640)
    self.tx6.config(text = "Prof. Manan M. Nanavati", font = ("Courier",20,"bold"))

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

```