DAYANANDA SAGAR UNIVERSITY

KUDLU GATE, BANGALORE – 560068

Bachelor of Technology
in
COMPUTER SCIENCE AND ENGINEERING

UG RESEARCH PROJECT-I/PRODUCT DEVELOPMENT FOUNDATION-I REPORT

TWO WAY SIGN LANGUAGE RECOGNITION SYSTEM FOR HEARING/ SPEECH IMPAIRED PEOPLE

By
Chethan Rao S – ENG19CS0076.

Under the supervision of
Dr. Sindhu P Menon
Professor, Dept. of CSE

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

(2021-2022)

DAYANANDA SAGAR UNIVERSITY

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

CERTIFICATE

This is to certify that the Phase-II project work titled **“TWO WAY SIGN LANGUAGE RECOGNITION SYSTEM FOR HEARING/ SPEECH IMPAIRED PEOPLE”** is carried out by **Chethan Rao S (ENG19CS0076)** bonafide student of Bachelor of Technology in Computer Science and Engineering at the School of Engineering, Dayananda Sagar University, Bangalore in partial fulfillment for the award of degree in Bachelor of Technology in Computer Science and Engineering, during the year **2021-2022**.

Dr. Sindhu P Menon	**Dr Girisha G S**	**Dr. A Srinivas**
Professor	Chairman CSE	Dean
Dept. of CS&E,	School of Engineering	School of Engineering
School of Engineering	Dayananda Sagar	Dayananda Sagar
Dayananda Sagar University	University	University
Date:	Date:	Date:

Name of the Examiner **Signature of Examiner**

1.

2.

DECLARATION

I, **Chethan Rao S (ENG19CS0076),** student of the sixth semester B.Tech in **Computer Science and Engineering**, at School of Engineering, **Dayananda Sagar University**, hereby declare that the UG RESEARCH PROJECT-I/PRODUCT DEVELOPMENT FOUNDATION-I titled **"TWO WAY SIGN LANGUAGE RECOGNITION SYSTEM FOR HEARING/ SPEECH IMPAIRED PEOPLE"** has been carried out by me and submitted in partial fulfillment for the award of degree in **Bachelor of Technology in Computer Science and Engineering** during the academic year **2021-2022**.

Student	**Signature**
Name : Chethan Rao S	
USN : ENG19CS0076	

Place : Bangalore

Date :

ACKNOWLEDGEMENT

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

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

We would like to thank **Dr. A Srinivas. Dean**, **School of Engineering & Technology**, **Dayananda Sagar University** for his constant encouragement and expert advice. It is a matter of immense pleasure to express our sincere thanks to **Dr. Girisha G S, Department Chairman**, **Computer Science, and Engineering**, **Dayananda Sagar University,** for providing the right academic guidance that made our task possible.

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

We would like to thank our **UG Research Project Coordinator Dr. Jayavrinda Vrindavanam** and all the staff members of Computer Science and Engineering for their support.

We are also grateful to our family and friends who provided us with every requirement throughout the course. We would like to thank one and all who directly or indirectly helped us in the Research work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abstract

Communication with the people having a hearing impairment is a difficult task. Signing has aided people with hearing and speech problems in expressing themselves to the rest of the world. Unfortunately, normal people do not understand their sign language. This creates a communication barrier between the hearing disability people and the normal people which should be broken. This paper proposes the recognition of American sign language gestures using an artificial intelligence tool, convolutional neural networks (CNN). This project is based on the vision-system. The system is capable of detecting the signing gestures from the video frames captured through the image. The vision-based system provides an easy and non-complex way for a person to interact with the computer. Initially the web-cam starts and captures the images of hand gestures. Then the captured image is supplied to the CNN model to predict. Finally, the predicted gesture is shown on the screen. Before the captured image is given to the CNN model, it will be applied with various pre-processing steps which includes techniques like converting images from RGB to HSV color space and mask operation. The proposed model can accurately recognize ten American Sign Language gesture alphabets.

CHAPTER 1

INTRODUCTION

CHAPTER 1 INTRODUCTION

Gestural(Sign) language is a way to communicate through hands and other body parts. As of February 2021, WHO estimates that 466 million people have hearing disabilities and frequently use sign language to communicate [5]. However, not everyone is conversant with sign language. It is also not an international language. This creates an impediment to communication between the deaf people and most auditory. Written communication is an option, however the Deaf community is often less adept in writing a spoken language. In addition, in one-on-one interactions, this sort of communication is slow. Thus, the fundamental objective of this study is to make progress in the field of sign language recognition.

Gestural recognition makes it possible for a computer to comprehend human actions and serves as a translator between the two. People frequently use sign language when they don't want to speak, but for the deaf and dumb, it is their sole means of communication. Hence dead and dumb people make use of sign language all across the world. They tend to use their regional form like ISL, ASL. In this work, the focus is on American Sign Language (ASL). 10 alphabets A, B, C, G, I, L, O, R, W and Y are considered here for classification.

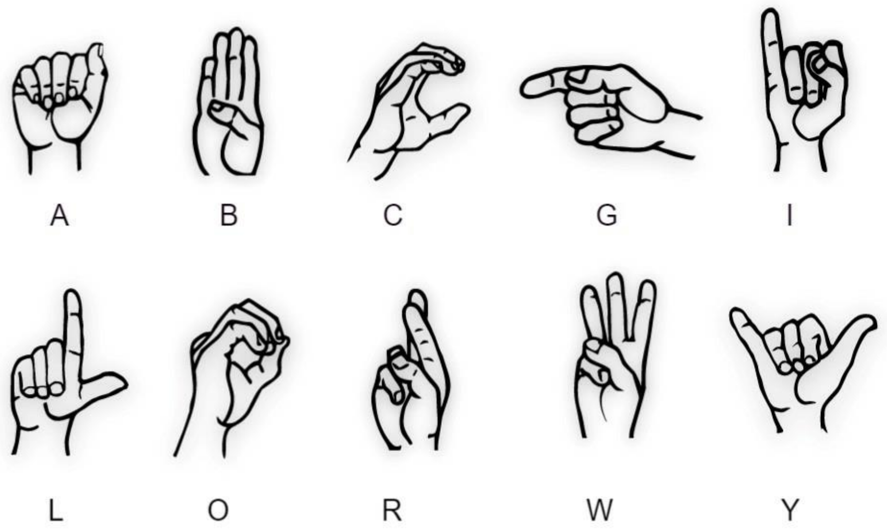


Fig 1.1: Finger Spelling American Sign Language

For identifying sign language gestures, Convolutional Neural Network (CNN) is employed. Once the CNN model is trained, it will be able to predict the corresponding alphabet of a sign language gesture. Instead of using any hand gloves, the proposed system works fine with the bare hands. Gestures will be recognized from the images captured from webcam. Then, using computer vision techniques and CNN, certain characteristics are extracted and classified. Convolutional Neural Networks (CNN) is used to identify images of sign language gestures. The reason for selecting CNN relative to other classifiers is that convolutional networks are faster in image extraction and classification.

CHAPTER 2

PROBLEM DEFINITION

CHAPTER 2 PROBLEM DEFINITION

Dumb people use hand signs to communicate, hence normal people face problem in recognizing their language by signs made. Hence there is a need of the systems which recognizes the different signs and conveys the information to the normal people. Also, there is a need for two-way communication which makes the system more interactive.
Hence the problem statement "**Two way sign language recognition system for hearing/ speech impaired people**" is proposed.

CHAPTER 3
LITERATURE REVIEW

CHAPTER 3 LITERATURE REVIEW

Herath, H.C.M. et. al.[1] had proposed a system for recognizing the gestures in Sinhala language in which they used a low-cost approach for processing the image. Their proposed system was different from other systems in terms of data collection. They had used a green background to capture photographs so that the green colour could be readily eliminated from the RGB colour space during processing. Then they converted the image to black and white. They used centroid method for classifying the mapping the hand signs. This method was able to map the hand gesture with its corresponding symbol regardless of hand size and position. Their model achieved an accuracy of 92%. Ronchetto, Franco [3] had proposed a way for recognizing the hand gesture of Argentinian sign language. In their proposed system, they first develop a library of hand gestures for the Argentinian Sign Language, and then they use ProbSom, a supervised adaption of self-organizing maps, for image processing descriptor extraction and subsequent handshape categorization. They also compared their technique with other algorithms like Random Forests, Support Vector Machines (SVM) and Neural Networks. But their technique of ProbSom based neural classifier had achieved an accuracy of above 90%. Recognizing continuous movements from sign language gestures is a difficult research problem. A gradient-based key frame extraction method was used by Tripathi, Kumud, and Neha Baranwal GC Nandi [2] to solve this challenge. These keyframes were useful for breaking down continuous sign language movements into a series of signs and deleting uninformative frames. Following the division of gestures, each sign was viewed as a separate gesture. Then, using Orientation Histogram (OH), features from pre-processed motions were retrieved, with Principal Component Analysis (PCA) used to reduce the dimension of features acquired after OH. Experiments were conducted on their own continuous ISL dataset, which was developed in the Robotics and Artificial Intelligence Laboratory (IIITA) using a Canon EOS camera. Different sorts of classifiers were used to test the probes, including Euclidean distance, correlation, Manhattan distance, city block distance, and so on. With several forms of distance classifiers, a comparative examination of their proposed scheme was performed. They discovered that the findings generated from correlation and Euclidean distance provide superior accuracy than other classifiers. Ahmed Elhagry, Rawan Glalal Elrayes [4] proposed using a vision-based method to convert some Egyptian Sign Language movements to isolated words. For categorization, they used two distinct architectures: Inception v3 CNN and Inception v3 CNN-LSTM. Using the first architecture, they were able to achieve a 90% accuracy rate. While the accuracy of the Inception v3 CNN-LSTM architecture was 72 percent. This demonstrated that CNN could recognize isolated sign languages with excellent accuracy.

CHAPTER 4

PROJECT DESCRIPTION

CHAPTER 4 PROJECT DESCRIPTION

The proposed system implements the sign language recognition using CNN. It has been designed for the two-way communication. Dataset is created on our own for 10 different alphabets. Initially server starts running and the client connects to the server using IP and port. After the connection is established, both the system starts receiving the video frames. I have used socket programming to implement the server-client architecture. The webcam captures the image of the gesture from the bounding box created on the screen. Then the image undergoes pre-processing stages like cropping, conversion of RGB to HSV color space and finally applying mask operation on it. Then the masked image is given to the CNN model for features extraction. Finally, the predicted gesture will be displayed on the screen.

CHAPTER 5
REQUIREMENTS

CHAPTER 5 REQUIREMENTS

Software:

Operating System : Windows 7/8/10

Language : Python 3.7

Tools : Tensorflow, opencv, numpy, keras.

Hardware:

Webcam : Min 3MB

Processor : Intel Core i5

RAM : Min 6GB

Hard Disk : Min 50GB

CHAPTER 6

METHODOLOGY

CHAPTER 6 METHODOLOGY

6.1 Existing System v/s Proposed System

In this proposed system, the way of capturing the data is different from the existing systems as we capture only the fixed bounding box and the user need not worry about the other portion. Existing systems are built only as a one-way communication i.e., there are no interactive systems. The proposed system is built as a two-way communication wherein the server will be running, and the client can connect to the server and both the users can interact with each other. Also, the accuracy achieved is slightly better than other existing models.

6.2 Data Collection

The process of acquiring data from multiple sources in order to train the model is known as data collection. It is an integral part, as the whole process of classification is dependent on the data. In this proposed system, the way of collecting the data is different from other systems. On the capturing window, a small green box is setup where the person has to perform the sign language gesture and capture the image. The capturing windows is as shown in the Fig 6.1. This is an efficient way of capturing the image as the person need not worry about the portion outside the green box, because it will not be considered. Only the green box portion is captured and saved in the device. The captured image will be in an RGB format. I have collected 1750 images of each of the 10 alphabets for training and 250 for testing. So, the training and testing data is split in the ratio of 70:30.

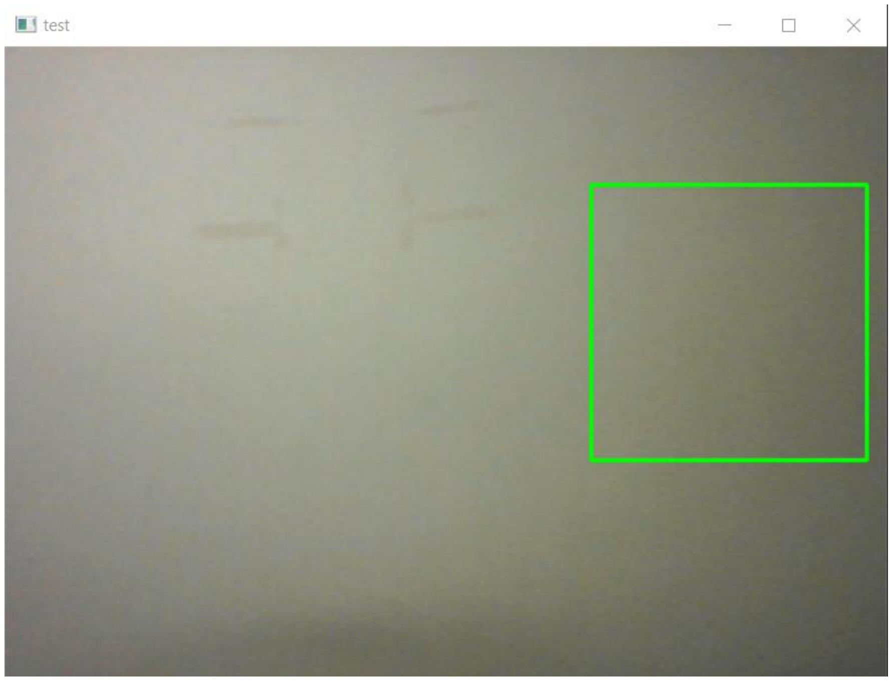


Fig 6.1: Image capturing window

6.3 Proposed Architecture

The proposed system is implemented as a two-way communication wherein a server will be running and the client can connect to the server. Initially, when the system begins running, the server starts and keeps on listening for accepting the connection. Once it receives the connection from the client, the webcam opens in both the server and client system and starts capturing the images. The image frames captured on the server side will be displayed on the client side and the image frames captured on the client side will be displayed on the server side. Then the captured image will undergo various pre-processing steps which include cropping the image, converting RGB to HSV color space and applying mask operation. The pre-processed image is then trained and classified using a Convolutional Neural Network (CNN). Finally, the predicted gesture from the model will be displayed on either of the systems. Fig 6.2 depicts the proposed system's whole flow.

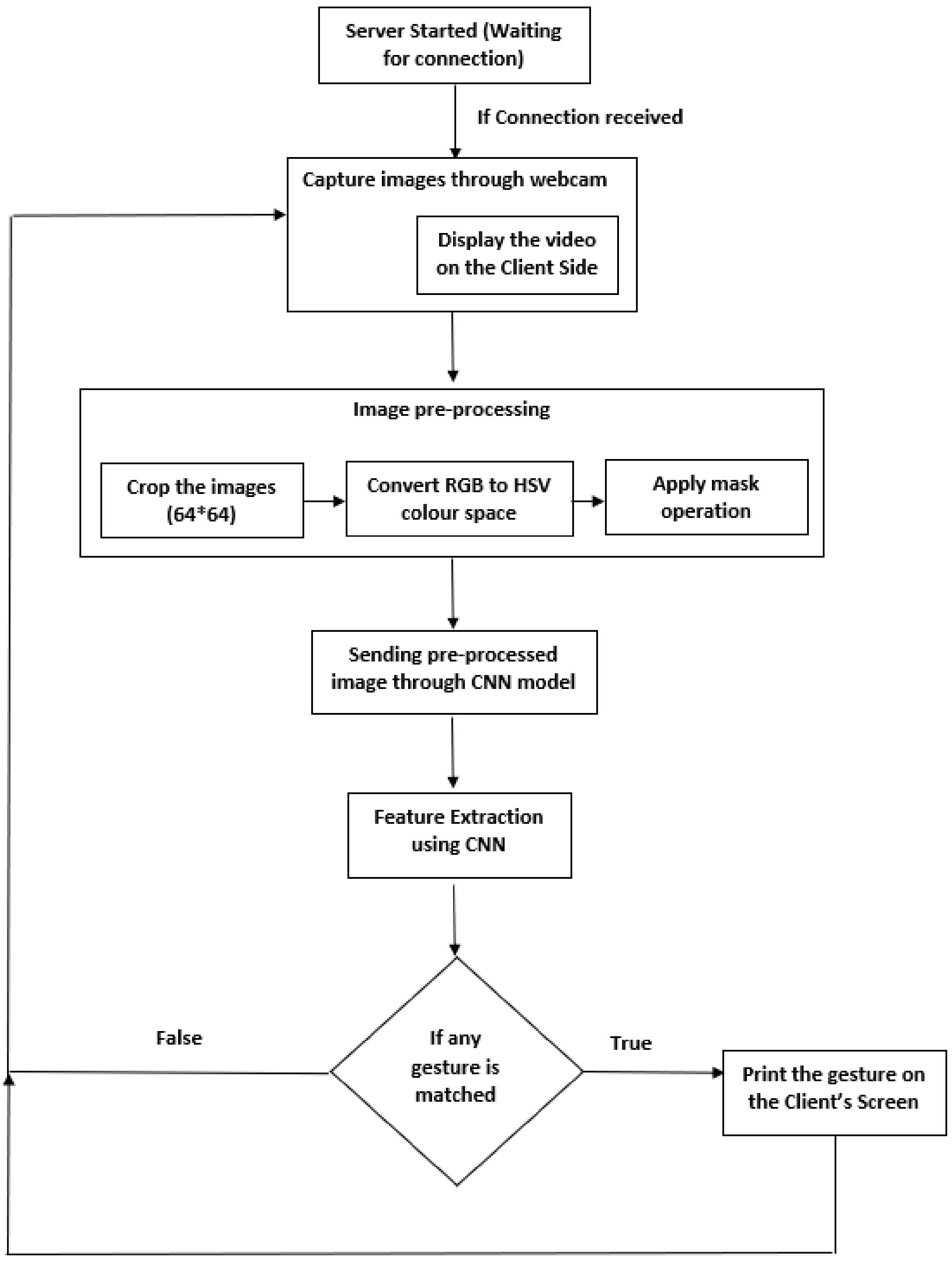

Server Started (Waiting for connection)
If Connection received
Capture images through webcam
Display the video on the Client Side
Image pre-processing
Crop the images (64*64)
Convert RGB to HSV colour space
Apply mask operation
Sending pre-processed image through CNN model
Feature Extraction using CNN
If any gesture is matched
False
True
Print the gesture on the Client's Screen

Fig 6.2 Flow diagram of sign language recognition system

The whole system can be abstractly viewed as two components, server and the client. The process of each component is as follows.

i) Server :

Servers are computers that are connected to a network and execute software that processes client requests and responds appropriately. This is the main component, as server is the one which provides the service to other systems. So, it has to be developed properly. In order to connect server to other client systems, I have used socket programming which is used to connect two or more systems on the same network with each other. A network socket is one end of a communication flow between two programs communicating via the internet. Server socket listens on a specific port at an IP address, while the client socket establishes a connection with the server.

Initially, we create a socket by calling the socket method from the socket module. Then we bind the socket to a specific IP address and port number. Then we make the socket to listen to the incoming connections from the network. Now a server socket is created and the client can connect to server on that specific IP and port. When the server system begins to execute, it will first listen to any connections from the network. As soon as the client connects to the server, Server accepts the connection, webcam starts capturing the video and then starts transferring the video frames to the client. The video frame will contain the person sitting before the webcam, his hand gesture in a bounding box and finally the predicted gesture in the form of text. Simultaneously, the video frames from the client will be received and displayed on the server system. Hence video frames are sent and received at the same time. The video frame will be first converted into a byte stream using the dump function in pickle module. Then it will be interpreted as a packed binary using the struct module. Then using the sendall(), we send the packed binary message(image frame) to the connected client. Now to receive the video frame from the client, we use the recv() function in the socket module and specify the length of the data to be received. Then unpack the data received (as it will be packed into binary data) using unpack() in struct module. Finally, we get the byte streams which will be loaded using the load() in pickle and we get the original image sent by the client.

ii) Client :

Clients are the computers that access the service made available by the server. These are the systems which try to connect to the server for communicating. To create a client system, we again create a socket. Then we ask the client user to specify the IP address and the port number to which they want to connect. After user specifies this information, connection is established with the server using the connect() in socket module. First of all, we receive the data from the server and then start sending the video to the server. This order has to be reversed in opposite system i.e., when one system (server) is sending the data , the other system (client) has to receive that data first before sending its data and vice versa. In the same way as explained in the server section, we send and receive the data in the client side as well.
The following are the steps followed after data collection.

6.3.1 Image Pre-Processing

Image captured through the webcam contains background noise. In order to remove the background noise, for adjusting the brightness, the contrast of the image, and cropping the image, the image has to be pre-processed. Pre-processed images give a better accuracy. The proposed system involves 3 steps of pre-processing. First, the captured image is cropped into a size of 64*64 as shown in Fig 6.3. Initially, the image will be in an RGB color space. The cropped image is converted into HSV color space as shown in Fig 6.4. Because HSV can separate pure colors from their lightness, it can be used to conduct many operations on the color (Hue) or the intensity of the color (Value). In fact, in most of the applications lightness is considered a noise. So, it is necessary that it has to be removed. Then I have set up a trackbar for adjusting the HSV values. Finally, mask is applied on the image to highlight the required portion as shown in Fig 6.5.

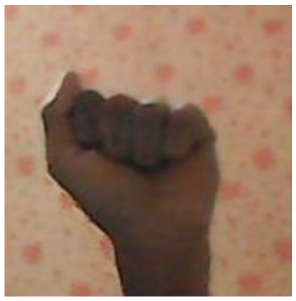


Fig 6.3: Image captured from webcam

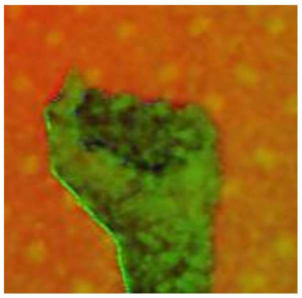


Fig 6.4: RGB to HSV converted image

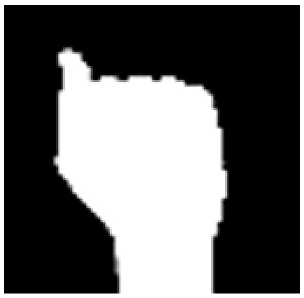


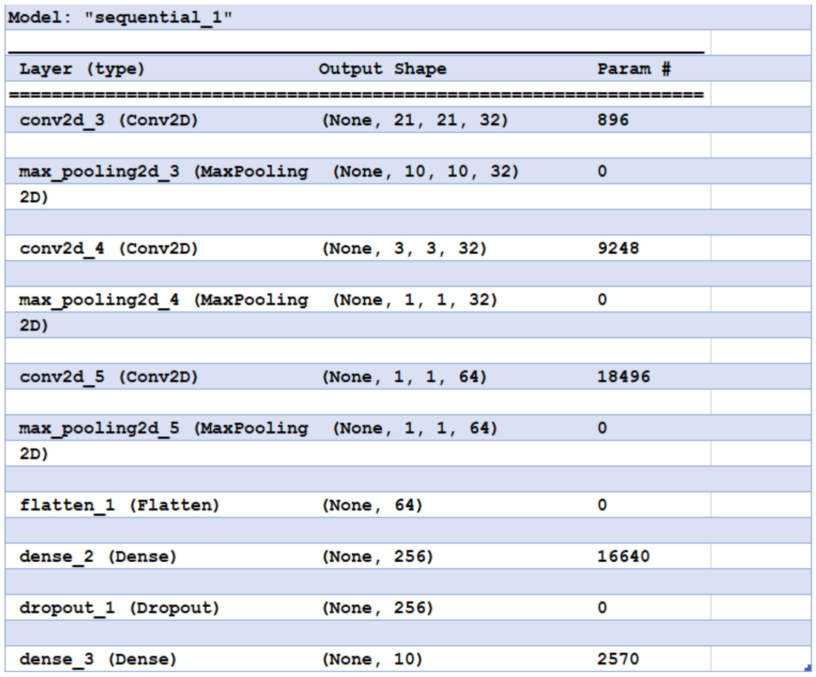
Fig 6.5: Image after applying mask

6.3.2 Feature Extraction

In image processing, feature extraction is a crucial step, because these are the ones that help the model to differentiate between the images. In this proposed system, for feature extraction, a convolutional neural network (CNN) is used. A CNN model is used to extract features from visual frames and predict hand gestures. It is a multi-layer neural network. It is majorly used in image processing. Input layer, three convolutional layers, three max pooling layers, one dense and one SoftMax output layer make up the model. The suggested CNN architecture employs three convolutional layers with varying window sizes, an activation function, and a rectified linear unit. Mean pooling, max pooling and stochastic pooling were all examined, with max pooling proving to be the best fit for our application. First convolutional layer consists of 32 filters of size 3*3 with activation function as

Rectified Linear Unit (ReLU). This layer extracts the important features from the image and passes them on to the next layer i.e., the pooling layer which decreases the activation map of the previous layer. Second convolution layer also consists of 32 filters of size 3*3. Third layer consists of 64 filters of size 3*3. Three layers of max pooling are used in the feature representation. To minimize significant information loss in feature representation, just two layers of pooling are launched. Then the flattening layer is added to convert to 1D. The classification stage uses dense, which are followed by activation functions. Dropout of 0.5 is added to prevent model from Overfitting. In classification, Softmax function is used. The hyper parameter tuning was done with Softmax (activation function), the epoch number was 25 and the batch size was 32 to suit the memory. Table 1 depicts the summary of CNN model.

Table 6.1: CNN model



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 21, 21, 32)        896

 max_pooling2d_3 (MaxPooling  (None, 10, 10, 32)       0
 2D)

 conv2d_4 (Conv2D)           (None, 3, 3, 32)          9248

 max_pooling2d_4 (MaxPooling  (None, 1, 1, 32)         0
 2D)

 conv2d_5 (Conv2D)           (None, 1, 1, 64)          18496

 max_pooling2d_5 (MaxPooling  (None, 1, 1, 64)         0
 2D)

 flatten_1 (Flatten)         (None, 64)                0

 dense_2 (Dense)             (None, 256)               16640

 dropout_1 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 10)                2570
```

The number of neurons in the last layer will be equal to the number of classes, hence the final layer has ten neurons. Finally, model is compiled using category cross-entropy as loss function and SGD as the optimizer.

6.3.3 Predicting the gesture

After feature extraction is done by the CNN, it classifies the image based on those features. After classifying, it will return the alphabet corresponding to the matched features. Finally, we print the returned alphabet in the form of text on the screen as shown in Fig 6.6

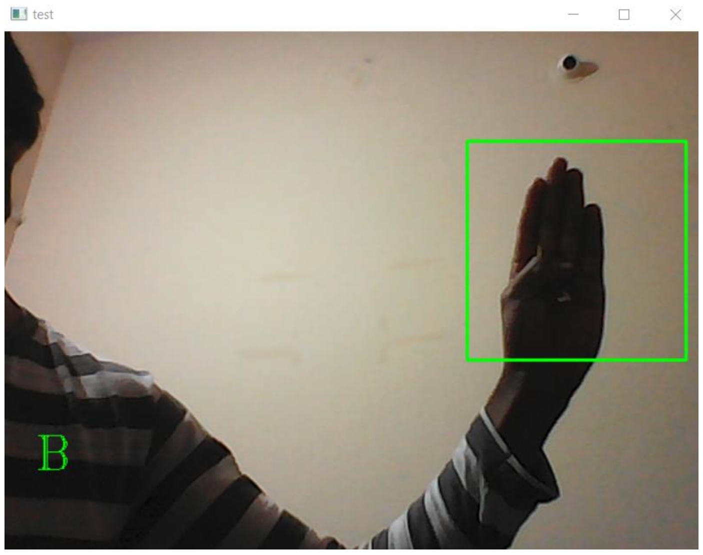


Fig 6.6 : Predicted gesture displayed in the form of text

CHAPTER 7
EXPERIMENTATION

CHAPTER 7 EXPERIMENTATION

In this proposed system, the way the dataset is created is different from other systems. Initially, I had setup a green bounding box on the screen by specifying (x1,y1) and (x2,y2) coordinates. Now, when the webcam starts, bounding box is displayed on the screen. I made the actor to sit in front of the webcam and show the gesture inside the bounding box. Further, I cropped only the bounding box section and sent it for pre-processing. This is an efficient way to capture the data, as there will not be much background noise. And also the complexity of detecting the hand before predicting the gesture will increase. So the proposed system has a simpler architecture.

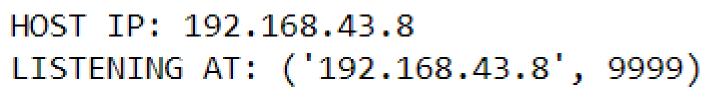
Initially, the testing accuracy of the model was fluctuating. So I added a 50% dropout before the softwax layer for removing the overfitting. Then the model performed well on the test data.

The optimizer used in this work is sgd . By analysis, it was observed that when compared with adam, sgd is more locally unstable and is more likely to converge to the minima at the flat or asymmetric basins/valleys which often have better generalization performance over other type minima. So my results can explain the better generalization performance of sgd over adam.

CHAPTER 8
RESULTS AND ANALYSIS

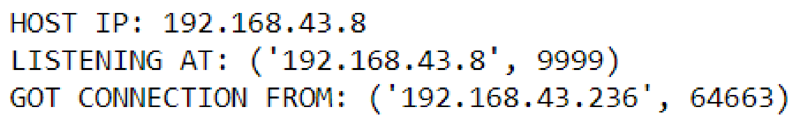
CHAPTER 8 RESULTS AND ANALYSIS

The proposed model is implemented as a two-way communication system which contains server and client. Server initially starts running on a specific port and waits for a client system to connect. When the client connects to the server on the specified IP and port, the server prints the IP of the connected system. The screen on the server system after executing looks as shown in Fig 8.1. and the server after receiving the connection looks shown in Fig 8.2



```
HOST IP: 192.168.43.8
LISTENING AT: ('192.168.43.8', 9999)
```

Fig 8.1 : Server running.



```
HOST IP: 192.168.43.8
LISTENING AT: ('192.168.43.8', 9999)
GOT CONNECTION FROM: ('192.168.43.236', 64663)
```

Fig 8.2 : Server after receiving connection.

Once the client connects to the server, the client starts transferring its video frames to the server which looks as shown in Fig 8.3.

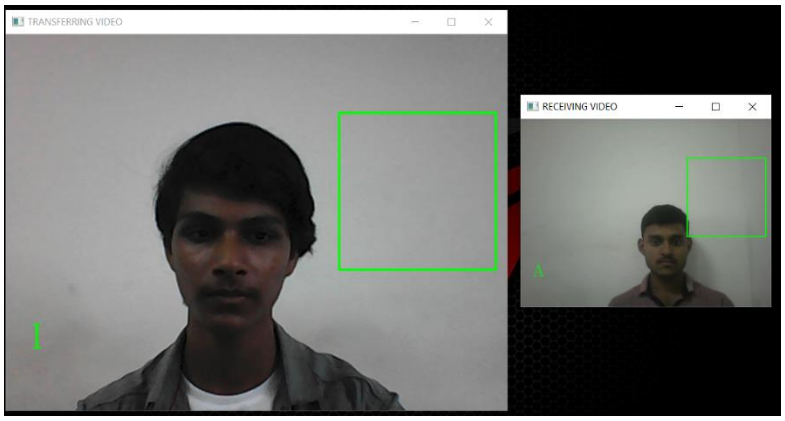
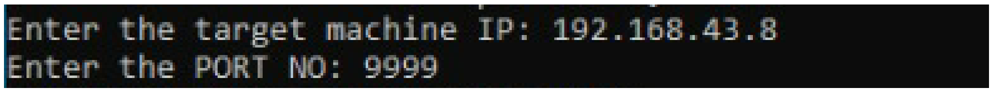


Fig 8.3 : Server (left image) sending video frames to client and receiving video frames from client (right image)

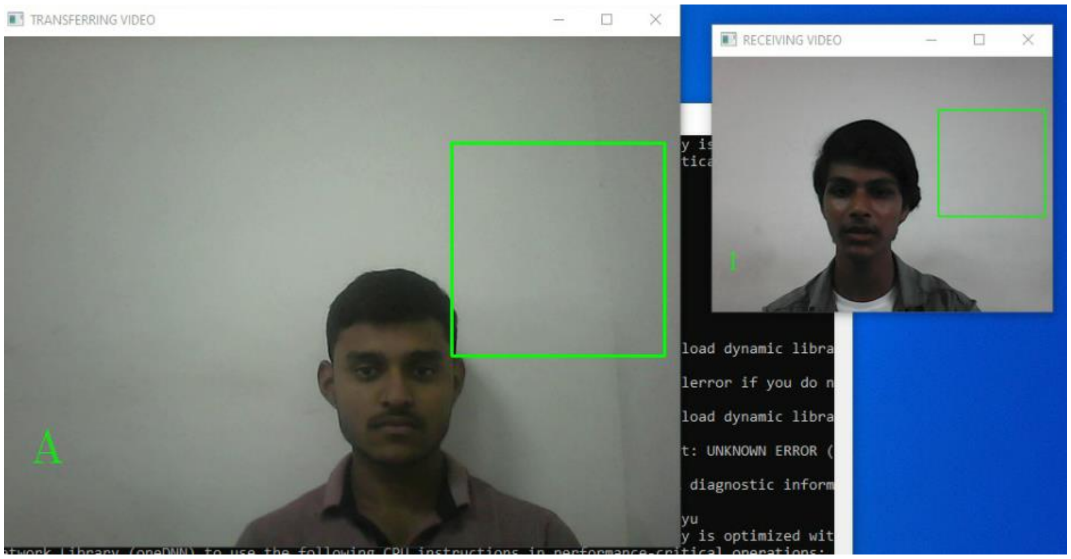
On the client side, the user will have to first enter the IP address and the port number in order to connect to the server. The screen on the client system after executing looks as shown in Fig 8.4.



```
Enter the target machine IP: 192.168.43.8
Enter the PORT NO: 9999
```

Fig 8.4 : Client system requesting for the IP address and port number.

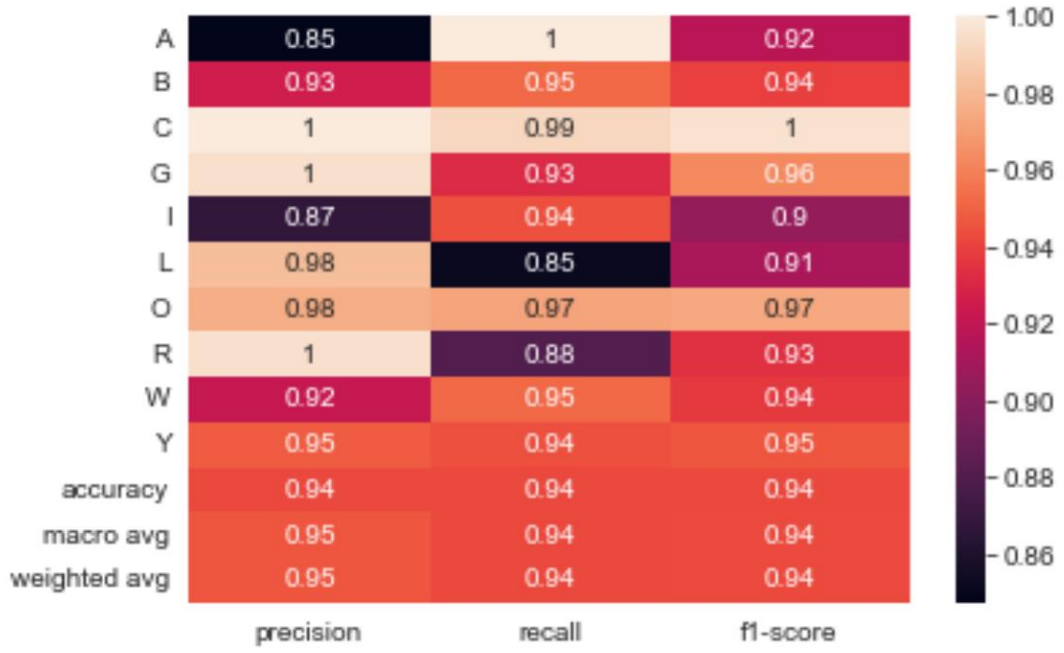
Simultaneously, the client starts receiving the video frames from the server which looks as shown in Fig 8.5.



```
y is
tica
load dynamic libra
lerror if you do n
load dynamic libra
t: UNKNOWN ERROR (
 diagnostic inform
yu
y is optimized wit
twork Library (oneDNN) to use the following CPU instructions in performance-critical operations:
```

Fig 8.5 : Client (left image) sending video frames to client and receiving video frames from server (right image)

The proposed model is evaluated based on 10 alphabetic American sign language: A, B, C, G, I, L, O, R, W and Y. By using deep learning technique i.e., Convolutional Neural Network (CNN), the model performed well. The number of images for each alphabet in the training set was 1750 whereas, in the testing set, it was 250 each. I have used a total of 17500 images to train the CNN model. The dataset is split in the ratio of 70:30 for training and testing respectively. An average accuracy of 94.2% is obtained. Precision, recall and f1-score of each class is shown in the Fig 8.6.



	precision	recall	f1-score
A	0.85	1	0.92
B	0.93	0.95	0.94
C	1	0.99	1
G	1	0.93	0.96
I	0.87	0.94	0.9
L	0.98	0.85	0.91
O	0.98	0.97	0.97
R	1	0.88	0.93
W	0.92	0.95	0.94
Y	0.95	0.94	0.95
accuracy	0.94	0.94	0.94
macro avg	0.95	0.94	0.94
weighted avg	0.95	0.94	0.94

1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86

Fig 8.6: Performance parameters for Classificiation

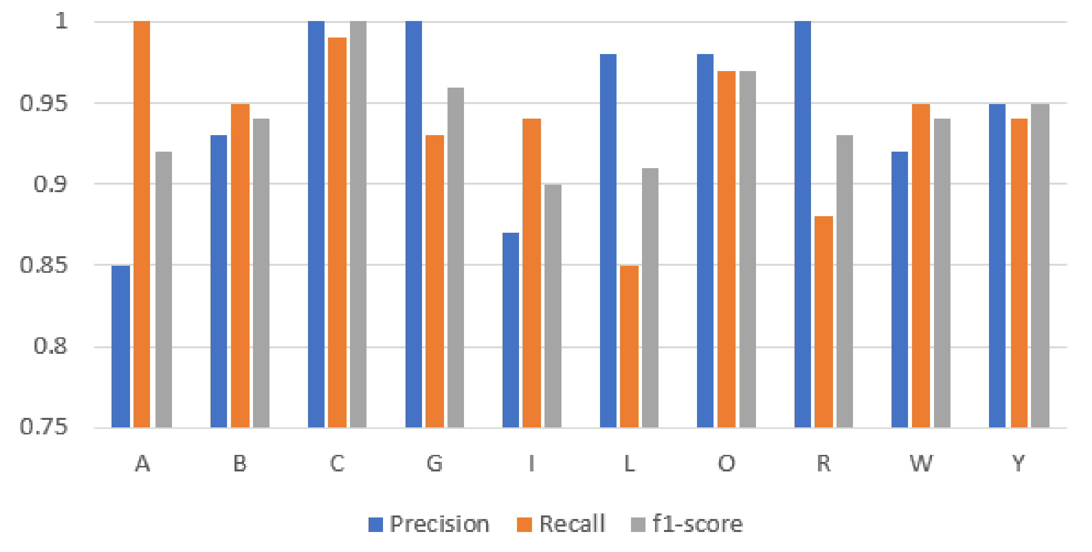


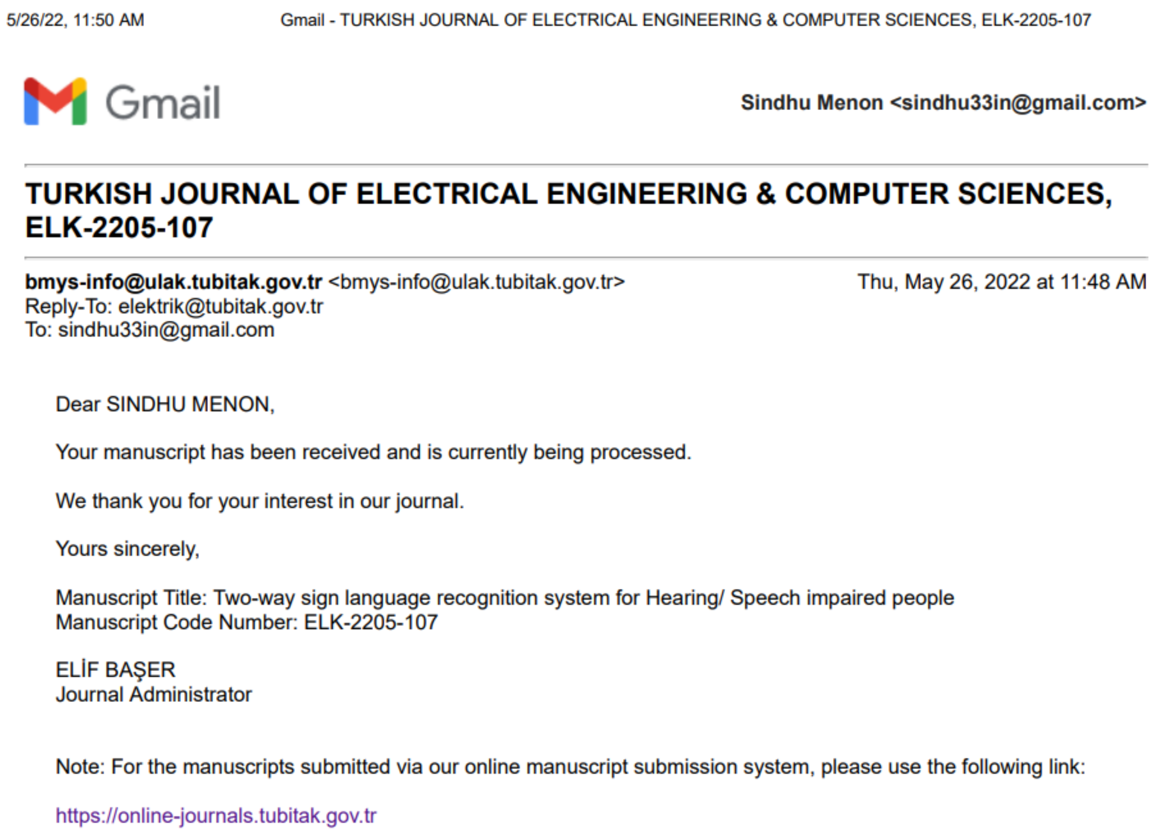
Fig 8.7: Precision, Recall and F1-score of each alphabet

CHAPTER 9

PAPER PUBLICATION /PATENT

CHAPTER 9 PAPER PUBLICATION/PATENT

- **Submitted a paper for publication to Turkish Journal of Electrical Engineering and Computer Sciences**



5/26/22, 11:50 AM Gmail - TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES, ELK-2205-107

Gmail **Sindhu Menon <sindhu33in@gmail.com>**

TURKISH JOURNAL OF ELECTRICAL ENGINEERING & COMPUTER SCIENCES, ELK-2205-107

bmys-info@ulak.tubitak.gov.tr <bmys-info@ulak.tubitak.gov.tr> Thu, May 26, 2022 at 11:48 AM
Reply-To: elektrik@tubitak.gov.tr
To: sindhu33in@gmail.com

Dear SINDHU MENON,

Your manuscript has been received and is currently being processed.

We thank you for your interest in our journal.

Yours sincerely,

Manuscript Title: Two-way sign language recognition system for Hearing/ Speech impaired people
Manuscript Code Number: ELK-2205-107

ELİF BAŞER
Journal Administrator

Note: For the manuscripts submitted via our online manuscript submission system, please use the following link:

https://online-journals.tubitak.gov.tr

CONCLUSION AND FUTURE WORK

The proposed method employs CNN for gesture training and classification. More useful features from the images are obtained and used for classification and training. For each sign, a total of 1750 static photos are utilized in the training process to ensure accuracy. Finally, the alphabet corresponding to the detected sign is displayed in text format. This project can be further extended by adding more alphabets so that the model recognizes more gestures. It can also be extended to sentence-level sign language with better accuracy.

REFERENCES

[1] Herath, H.C.M. & W.A.L.V.Kumari, & Senevirathne, W.A.P.B & Dissanayake, Maheshi. “Image Based Sign Language Recognition System For Sinhala Sign Language “ (2013)

[2] Tripathi, Kumud, and Neha Baranwal GC Nandi. "Continuous Indian Sign Language Gesture Recognition and Sentence Formation.", Procedia Computer Science 54 (2015): 523531.

[3] Ronchetti, Franco, Facundo Quiroga, César Armando Estrebou, and Laura Cristina Lanzarini. "Handshape recognition for argentinian sign language using probsom." Journal of Computer Science & Technology 16 (2016).

[4] Ahmed Elhagry , Rawan Glalal Elrayes ,(2021) ,”Egyptian Sign Language Recognition Using CNN and LSTM” (2021)

[5] “Deafness and hearing loss,” World Health Organization. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/deafnessand-hearing-loss.

[6] V. Athitsos, Quan Yuan and S. Sclaroff, “A Unified Framework for Gesture Recognition and Spatiotemporal Gesture Segmentation”, IEEE Transactions, vol. 31, pp. 1685–1699, September (2009).

[7] A. Saxena, D. K. Jain and A. Singhal, “Sign Language Recognition using Principal Component Analysis”, Fourth International Conference on IEEE Communication Systems and Network Technologies (CSNT) 2014, pp. 810–813, 7–9 April (2014).

[8] D. Mazumdar, A. K. Talukdar and K. K. Sarma, “Gloved and Free Hand Tracking based Hand Gesture Recognition”, IEEE, 2013, pp. 197–202, 13–14 September (2013).

[9] A. S. Elons ; Howida A. Shedeed ; M. F. Tolba ”Arabic sign language recognition with 3D convolutional neural networks” 2017

[10] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei, “Large-scale video classification with convolutional neural networks,” in CVPR, 2014.

[11] N. Purva, K. Vaishali, “Indian Sign language Recognition”: A Review, IEEE proceedings on International Conference on Electronics and Communication Systems, pp. 452-456, 2014

[12] Noor Tubaiz, Tamer Shanableh, and Khaled Assaleh, "Glove-Based Continuous Arabic Sign Language Recognition in User-Dependent Mode," IEEE Transactions on Human-Machine Systems, Vol. 45, NO. 4, August 2015.

[13] L. Rioux-Maldague and P. Giguere, “Sign language fingerspelling classification from depth and color images using a deep belief network,” in Computer and Robot Vision (CRV), 2014 Canadian Conference on, pp. 92–97, IEEE, 2014.

[14] H. Cooper, E.-J. Ong, N. Pugeault, and R. Bowden, “Sign language recognition using sub-units,” Journal of Machine Learning Research, vol. 13, pp. 2205–2231, Jul 2012.

[15] Huang, J., Zhou, W., & Li, H. (2015). “Sign Language Recognition using 3D convolutional neural networks”. IEEE International Conference on Multimedia and Expo (ICME) (pp. 1-6). Turin: IEEE.

[16] M. Geetha and U. C. Manjusha, , “A Vision Based Recognition of Indian Sign Language Alphabets and Numerals Using B-Spline Approximation”, International Journal on Computer Science and Engineering (IJCSE), vol. 4, no. 3, pp. 406-415. 2012.

[17] Dong, C., Leu, M. C., & Yin, Z. (2015). “American sign language alphabet recognition using microsoft kinect”. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops (pp. 44-52).

[18] Pigou, L., Dieleman, S., Kindermans, P. J., & Schrauwen, B. (2014, September). “Sign language recognition using convolutional neural networks”. In European Conference on Computer Vision (pp. 572-578). Springer, Cham.

[19] Tang, A., Lu, K., Wang, Y., Huang, J., & Li, H. (2015). “A real-time hand posture recognition system using deep neural networks”. ACM Transactions on Intelligent Systems and Technology (TIST), 6(2), 1-23.

[20] Cooper, H., Ong, E.J., Pugeault, N., Bowden, R.: “Sign language recognition using sub-units”. The Journal of Machine Learning Research 13(1), 2205–2231 (2012)

CODE

1. capture.py

```
import cv2
import time
import numpy as np
import os

def nothing(x):
    pass

image_x, image_y = 64, 64

def create_folder(folder_name):
    if not os.path.exists('./mydata/training_set/' + folder_name):
        os.mkdir('./mydata/training_set/' + folder_name)
    if not os.path.exists('./mydata/test_set/' + folder_name):
        os.mkdir('./mydata/test_set/' + folder_name)




def capture_images(ges_name):
    create_folder(str(ges_name))
    cam = cv2.VideoCapture(0)
    cv2.namedWindow("test")
    img_counter = 0
    t_counter = 1
    training_set_image_name = 1
    test_set_image_name = 1
```

```
listImage = [1,2,3,4,5]
flag=0

cv2.namedWindow("Trackbars")

cv2.createTrackbar("L - H", "Trackbars", 0, 179, nothing)
cv2.createTrackbar("L - S", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("L - V", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("U - H", "Trackbars", 179, 179, nothing)
cv2.createTrackbar("U - S", "Trackbars", 255, 255, nothing)
cv2.createTrackbar("U - V", "Trackbars", 255, 255, nothing)

for loop in listImage:
    while True:
        ret, frame = cam.read()
        frame = cv2.flip(frame, 1)
        l_h = cv2.getTrackbarPos("L - H", "Trackbars")
        l_s = cv2.getTrackbarPos("L - S", "Trackbars")
        l_v = cv2.getTrackbarPos("L - V", "Trackbars")
        u_h = cv2.getTrackbarPos("U - H", "Trackbars")
        u_s = cv2.getTrackbarPos("U - S", "Trackbars")
        u_v = cv2.getTrackbarPos("U - V", "Trackbars")

        img = cv2.rectangle(frame, (425, 100), (625, 300), (0, 255, 0), thickness=2,
lineType=8, shift=0)

        lower_blue = np.array([l_h, l_s, l_v])
        upper_blue = np.array([u_h, u_s, u_v])
        imcrop = img[102:298, 427:623]
        hsv = cv2.cvtColor(imcrop, cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(hsv, lower_blue, upper_blue)
        cv2.putText(frame, str(img_counter), (30, 400),
cv2.FONT_HERSHEY_TRIPLEX, 1.5, (127, 127, 255))
        cv2.imshow("test", frame)
        cv2.imshow("mask", mask)

        if cv2.waitKey(1) == ord('c'):
            if t_counter <= 350:
                img_name = "./mydata/training_set/" + str(ges_name) +
"/{}.png".format(training_set_image_name)
                save_img = cv2.resize(mask, (image_x, image_y))
                cv2.imwrite(img_name, save_img)
                print("{} written!".format(img_name))
                training_set_image_name += 1

            if t_counter > 350 and t_counter <= 400:
                img_name = "./mydata/test_set/" + str(ges_name) +
"/{}.png".format(test_set_image_name)
                save_img = cv2.resize(mask, (image_x, image_y))
                cv2.imwrite(img_name, save_img)
                print("{} written!".format(img_name))
                test_set_image_name += 1
                if test_set_image_name > 250:
                    break

            t_counter += 1
            if t_counter == 401:
                t_counter = 1
            img_counter += 1
            elif cv2.waitKey(1) == ord('q'):
                flag=1
                break

        if test_set_image_name > 250:
            break
        if flag==1:
            break
    cam.release()
    cv2.destroyAllWindows()

ges_name = input("Enter gesture name: ")
capture_images(ges_name)
```

2. cnn_model.py

```
# Part 1 - Building the CNN
#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
from keras import optimizers
import tensorflow as tf

# Initialing the CNN
classifier = Sequential()
```

```
# Step 1 - Convolutio Layer
classifier.add(Convolution2D(32, 3,  3, input_shape = (64, 64, 3), activation = 'relu'))

#step 2 - Pooling
classifier.add(MaxPooling2D(pool_size =(2,2)))

# Adding second convolution layer
classifier.add(Convolution2D(32, 3,  3, activation = 'relu'))
classifier.add(MaxPooling2D(pool_size =(2,2)))

#Adding 3rd Concolution Layer
classifier.add(Convolution2D(64, 3,  3, activation = 'relu',padding='same'))
classifier.add(MaxPooling2D(pool_size =(2,2),padding='same'))


#Step 3 - Flattening
classifier.add(Flatten())

#Step 4 - Full Connection
classifier.add(Dense(256, activation = 'relu'))
classifier.add(Dropout(0.5))
classifier.add(Dense(10, activation = 'softmax'))

#Compiling The CNN
classifier.compile(
            optimizer = tf.keras.optimizers.SGD(lr = 0.01),
            loss = 'categorical_crossentropy',
            metrics = ['accuracy'])

#Part 2 Fittting the CNN to the image
```

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

training_set = train_datagen.flow_from_directory(
        'mydata/training_set',
        target_size=(64, 64),
        batch_size=32,
        class_mode='categorical')

test_set = test_datagen.flow_from_directory(
        'mydata/test_set',
        target_size=(64, 64),
        batch_size=32,
        class_mode='categorical')

model = classifier.fit_generator(
        training_set,
        steps_per_epoch=500,
        epochs=25,
        validation_data = test_set,
        validation_steps = len(test_set)//32,
      )
# import h5py
# classifier.save('Trained_model.h5')
```

3. server.py

```
import cv2
import numpy as np
from keras.preprocessing import image
import tensorflow as tf
import socket, pickle,struct,imutils
import os

def nothing(x):
    pass

image_x, image_y = 64,64

from keras.models import load_model
classifier = load_model('Trained_model.h5')

try:
    os.mkdir('./temp')
except:
    pass

def predictor(img):
    cv2.imwrite(r".\temp\1.png",img)
```

```
img = image.load_img(r".\temp\1.png",target_size=(64,64))


test_image = image.img_to_array(img)

#       print(test_image.shape)

test_image = np.expand_dims(test_image, axis = 0)

# predict image using classifier

result = classifier.predict(test_image)


if result[0][0] == 1:

    return 'A'

elif result[0][1] == 1:

    return 'B'

elif result[0][2] == 1:

    return 'C'

elif result[0][3] == 1:

    return 'G'

elif result[0][4] == 1:

    return 'I'

elif result[0][5] == 1:

    return 'L'

elif result[0][6] == 1:

    return 'O'

elif result[0][7] == 1:

    return 'R'
    elif result[0][8] == 1:
        return 'W'
    elif result[0][9] == 1:
        return 'Y'



server_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
host_name  = socket.gethostname()
host_ip = socket.gethostbyname(host_name)
print('HOST IP:',host_ip)
port = 9999
socket_address = (host_ip,port)

# Socket Bind
server_socket.bind(socket_address)

# Socket Listen
server_socket.listen(5)
print("LISTENING AT:",socket_address)
data = b""
payload_size = struct.calcsize("Q")

client_socket,addr = server_socket.accept()
```

```
print('GOT CONNECTION FROM:',addr)
cam = cv2.VideoCapture(0)

cv2.namedWindow("Trackbars")

cv2.createTrackbar("L - H", "Trackbars", 0, 179, nothing)
cv2.createTrackbar("L - S", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("L - V", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("U - H", "Trackbars", 179, 179, nothing)
cv2.createTrackbar("U - S", "Trackbars", 255, 255, nothing)
cv2.createTrackbar("U - V", "Trackbars", 123, 255, nothing)

cv2.namedWindow("TRANSFERRING VIDEO")

img_counter = 0

img_text = ''
while True:
    ret, frame = cam.read()
    frame = cv2.flip(frame,1)
    l_h = cv2.getTrackbarPos("L - H", "Trackbars")
    l_s = cv2.getTrackbarPos("L - S", "Trackbars")
    l_v = cv2.getTrackbarPos("L - V", "Trackbars")
    u_h = cv2.getTrackbarPos("U - H", "Trackbars")
    u_s = cv2.getTrackbarPos("U - S", "Trackbars")
    u_v = cv2.getTrackbarPos("U - V", "Trackbars")


    img = cv2.rectangle(frame, (425,100),(625,300), (0,255,0), thickness=2, lineType=8,
shift=0)

    lower_blue = np.array([l_h, l_s, l_v])
    upper_blue = np.array([u_h, u_s, u_v])
    imcrop = img[102:298, 427:623]
#     print(imcrop.shape)
    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))
    cv2.imshow("TRANSFERRING VIDEO", frame)
    cv2.imshow("mask", mask)

#     print(np.count_nonzero(mask))

    if client_socket:
        send = imutils.resize(frame,width=320)
        a = pickle.dumps(send)

        message = struct.pack("Q",len(a))+a
        client_socket.sendall(message)

        while len(data) < payload_size:
            packet = client_socket.recv(4*1024) # 4K
            if not packet: break
            data+=packet
        packed_msg_size = data[:payload_size]
        data = data[payload_size:]
        try:
            msg_size = struct.unpack("Q",packed_msg_size)[0]
        except:
            break

        while len(data) < msg_size:
            data += client_socket.recv(4*1024)
        frame_data = data[:msg_size]
        data  = data[msg_size:]
        f = pickle.loads(frame_data)
        cv2.imshow("RECEIVING VIDEO",f)


    if cv2.waitKey(1)==ord('q'):
        server_socket.close()
        client_socket.close()
```

```
        print("EXITING from Server...")
        break

    if(np.count_nonzero(mask)<=10000):
        continue

    if mask.any():
        img_text = predictor(mask)
    else:
        continue

cam.release()
cv2.destroyAllWindows()
```

4. client.py

```
import cv2
import numpy as np
from keras.preprocessing import image
import tensorflow as tf
import socket,pickle,struct,imutils

def nothing(x):
    pass

image_x, image_y = 64,64
```

```
from keras.models import load_model
classifier = load_model('Trained_model.h5')

try:
    os.mkdir("./temp")
except:
    pass

def predictor(img):
    cv2.imwrite(r".\temp\1.png",img)
    img = image.load_img(r".\temp\1.png",target_size=(64,64))

    test_image = image.img_to_array(img)
    #       print(test_image.shape)
    test_image = np.expand_dims(test_image, axis = 0)
    # predict image using classifier
    result = classifier.predict(test_image)

    if result[0][0] == 1:
        return 'A'
    elif result[0][1] == 1:
        return 'B'
    elif result[0][2] == 1:
        return 'C'
    elif result[0][3] == 1:
        return 'G'
    elif result[0][4] == 1:
        return 'I'
    elif result[0][5] == 1:
        return 'L'
    elif result[0][6] == 1:
        return 'O'
    elif result[0][7] == 1:
        return 'R'
    elif result[0][8] == 1:
        return 'W'
    elif result[0][9] == 1:
        return 'Y'

cv2.namedWindow("Trackbars")

cv2.createTrackbar("L - H", "Trackbars", 0, 179, nothing)
cv2.createTrackbar("L - S", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("L - V", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("U - H", "Trackbars", 179, 179, nothing)
cv2.createTrackbar("U - S", "Trackbars", 255, 255, nothing)
cv2.createTrackbar("U - V", "Trackbars", 123, 255, nothing)

#cv2.namedWindow("test")

img_counter = 0

img_text = ''

# create socket
client_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
host_ip = input("Enter the target machine IP: ")
port = int(input("Enter the PORT NO: "))
client_socket.connect((host_ip,port)) # a tuple
data = b""
payload_size = struct.calcsize("Q")
```

```
vid = cv2.VideoCapture(0,cv2.CAP_DSHOW)
while True:
    ret, frame = vid.read()
    l_h = cv2.getTrackbarPos("L - H", "Trackbars")
    l_s = cv2.getTrackbarPos("L - S", "Trackbars")
    l_v = cv2.getTrackbarPos("L - V", "Trackbars")
    u_h = cv2.getTrackbarPos("U - H", "Trackbars")
    u_s = cv2.getTrackbarPos("U - S", "Trackbars")
    u_v = cv2.getTrackbarPos("U - V", "Trackbars")


    img = cv2.rectangle(frame, (425,100),(625,300), (0,255,0), thickness=2, lineType=8,
shift=0)

    lower_blue = np.array([l_h, l_s, l_v])
    upper_blue = np.array([u_h, u_s, u_v])
    imcrop = img[102:298, 427:623]
#    print(imcrop.shape)
    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))
    cv2.imshow("TRANSFERRING VIDEO", frame)
    cv2.imshow("mask", mask)


    while len(data) < payload_size:
        packet = client_socket.recv(4*1024) # 4K

        if not packet: break
        data+=packet
```

```
packed_msg_size = data[:payload_size]
data = data[payload_size:]
try:
    msg_size = struct.unpack("Q",packed_msg_size)[0]
    try:
        if(msg_size<2000):
            break
    except:
        pass
except:
    break

while len(data) < msg_size:
    data += client_socket.recv(4*1024)

frame_data = data[:msg_size]
data  = data[msg_size:]
receiving_frame = pickle.loads(frame_data)
cv2.imshow("RECEIVING VIDEO",receiving_frame)

try:
        f = imutils.resize(frame,width=320)
except:
    continue
a = pickle.dumps(f)
message = struct.pack("Q",len(a))+a
client_socket.sendall(message)



key = cv2.waitKey(1) & 0xFF
if key  == ord('q'):
        client_socket.close()
        break

    if(np.count_nonzero(mask)<=10000):
        continue

    if mask.any():
        img_text = predictor(mask)
    else:
        continue

vid.release()
cv2.destroyAllWindows()
```

5. recognize.py

```
import cv2
import numpy as np
from keras.preprocessing import image
import tensorflow as tf
import os

def nothing(x):
    pass

try:
    os.mkdir('./temp')
except:
    pass
```

```
image_x, image_y = 64,64

from keras.models import load_model
classifier = load_model('Trained_model.h5')

def predictor(img):
    cv2.imwrite(r".\temp\1.png",img)
    img = image.load_img(r".\temp\1.png",target_size=(64,64))

    test_image = image.img_to_array(img)
    #       print(test_image.shape)
    test_image = np.expand_dims(test_image, axis = 0)
    #       test_image = test_image.reshape((64,64,3))
    # predict image using classifier
    result = classifier.predict(test_image)

    if result[0][0] == 1:
        return 'A'
    elif result[0][1] == 1:
        return 'B'
    elif result[0][2] == 1:
        return 'C'
    elif result[0][3] == 1:
        return 'G'
    elif result[0][4] == 1:
        return 'I'
    elif result[0][5] == 1:
        return 'L'
    elif result[0][6] == 1:
        return 'O'
    elif result[0][7] == 1:
            return 'R'
        elif result[0][8] == 1:
            return 'W'
        elif result[0][9] == 1:
            return 'Y'

cam = cv2.VideoCapture(0)

cv2.namedWindow("Trackbars")

cv2.createTrackbar("L - H", "Trackbars", 0, 179, nothing)
cv2.createTrackbar("L - S", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("L - V", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("U - H", "Trackbars", 179, 179, nothing)
cv2.createTrackbar("U - S", "Trackbars", 255, 255, nothing)
cv2.createTrackbar("U - V", "Trackbars", 123, 255, nothing)

# cv2.namedWindow("test")

img_counter = 0

img_text = ''
while True:
    ret, frame = cam.read()
    frame = cv2.flip(frame,1)
    l_h = cv2.getTrackbarPos("L - H", "Trackbars")
    l_s = cv2.getTrackbarPos("L - S", "Trackbars")
    l_v = cv2.getTrackbarPos("L - V", "Trackbars")
    u_h = cv2.getTrackbarPos("U - H", "Trackbars")
    u_s = cv2.getTrackbarPos("U - S", "Trackbars")
    u_v = cv2.getTrackbarPos("U - V", "Trackbars")
    img = cv2.rectangle(frame, (425,100),(625,300), (0,255,0), thickness=2, lineType=8,
shift=0)

    lower_blue = np.array([l_h, l_s, l_v])
    upper_blue = np.array([u_h, u_s, u_v])
    imcrop = img[102:298, 427:623]
#     imcrop = cv2.resize(imcrop,(64,64))
#     print(imcrop.shape)
    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))
    cv2.imshow("test", frame)
    cv2.imshow("mask", mask)

#     mask = cv2.resize(mask, (64,64))
#     print(np.count_nonzero(mask))

    if cv2.waitKey(1)==ord('q'):
        break

    if(np.count_nonzero(mask)<=10000):
        continue

    if mask.any():
        img_text = predictor(mask)
    else:
        continue
```

```
cam.release()
cv2.destroyAllWindows()
```