Introduction

Motion of any body part like face, hand is a form of gesture. Here for gesture recognition we are using image processing and computer vision. Gesture recognition enables computer to understand human actions and also acts as an interpreter between computer and human. This could provide potential to human to interact naturally with the computers without any physical contact of the mechanical devices. Gestures are performed by deaf and dumb community to perform sign language. This community used sign language for their communication when broadcasting audio is impossible, or typing and writing is difficult, but there is the vision possibility. At that time sign language is the only way for exchanging information between people. Normally sign language is used by everyone when they do not want to speak, but this is the only way of communication for deaf and dumb community. Sign language is also serving the same meaning as spoken language does. This is used by deaf and dumb community all over the world but in their regional form like ISL, ASL. Sign language can be performed by using Hand gesture either by one hand or two hands. It is of two type Isolated sign language and continuous sign language. Isolated sign language consists of single gesture having single word while continuous ISL or Continuous Sign language is a sequence of gestures that generate a meaningful sentence.

In this report we performed isolated ASL gesture recognition technique. Sign Language Deaf people around the world communicate using sign language as distinct from spoken language in their everyday a visual language that uses a system of manual, facial and body movements as the means of communication. Sign language is not an universal language, and different sign languages are used in different countries, like the many spoken languages all over the world. Some countries such as Belgium, the UK, the USA or India may have more than one sign language. Hundreds of sign languages are in used around the world, for instance, Japanese Sign Language, British Sign Language (BSL), Spanish Sign Language, Turkish Sign Language.



**Finger Spelling American Sign Language**

**Algorithms and Method**

**Convolutional Neural Network (CNN)**

Neural networks, as its name suggests, is a machine learning technique which is modeled after the brain structure. It comprises of a network of learning units called neurons. These neurons learn how to convert input signals (e.g. picture of a cat) into corresponding output signals (e.g. the label “cat”), forming the basis of automated recognition.

A convolutional neural network (CNN, or ConvNet) is a type of feed­forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex.

CNNs have repetitive blocks of neurons that are applied across space (for images) or time (for audio signals etc). For images, these blocks of neurons can be interpreted as 2D convolutional kernels, repeatedly applied over each patch of the image. For speech, they can be seen as the 1D convolutional kernels applied across time windows. At training time, the weights for these repeated blocks are 'shared', i.e. the weight gradients learned over various image patches are averaged.

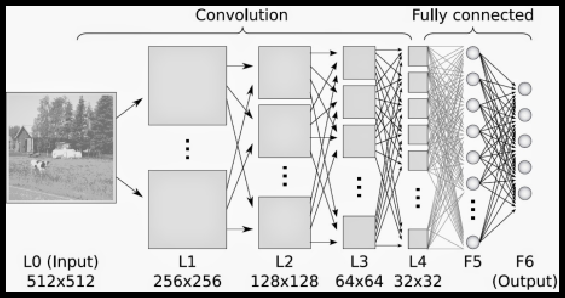
**CNN Summarized in 4 Steps**

There are four main steps in CNN:

•**convolution**

**•subsampling**

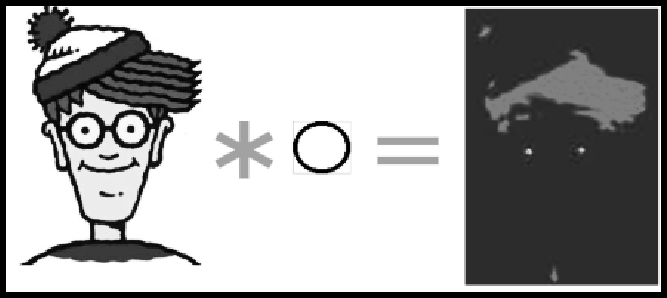
**•activation**

**•full connectedness**

**Convolutional neural network**

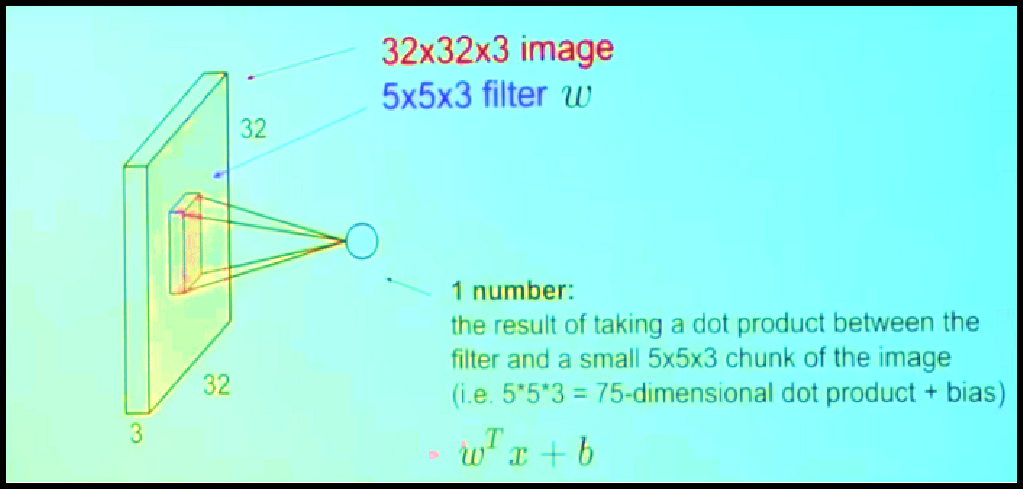
**Convolution**

The first layers that receive an input signal are called convolution filters. Convolution is a process where the network tries to label the input signal by referring to what it has learned in the past. If the input signal looks like previous cat images it has seen before, the “cat” reference signal will be mixed into, or convolved with, the input signal. The resulting output signal is then passed on to the next layer.

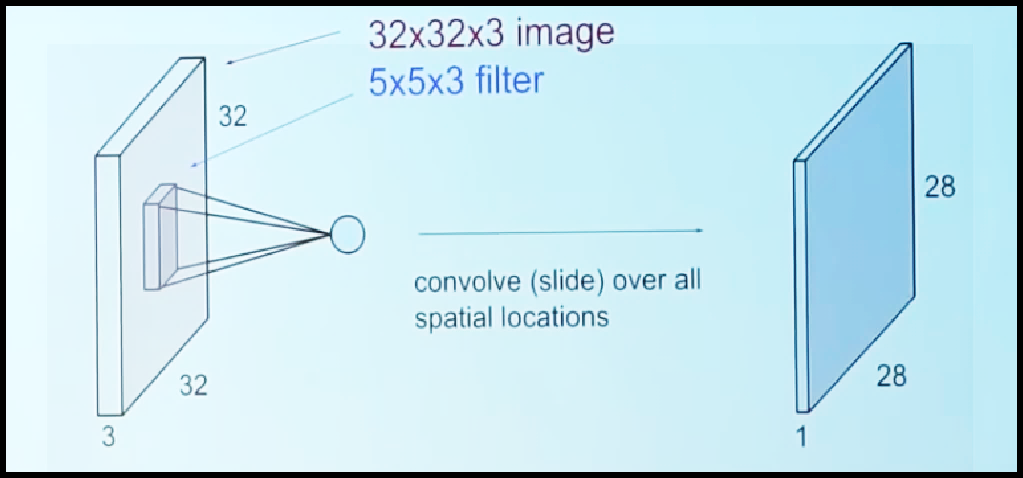


Convolving Wally with a circle filter. The circle filter responds strongly to the eyes.

Convolution has the nice property of being **translational invariant**. Intuitively, this means that each convolution filter represents a feature of interest (e.g whiskers, fur), and the CNN algorithm learns which features comprise the resulting reference (i.e. cat). The output signal strength is not dependent on where the features are located, but simply whether the features are present. Hence, a cat could be sitting in different positions, and the CNN algorithm would still be able to recognize it.

For e.g. suppose we convolve a 32x32x3 (32x32 image with 3 channels R, G and B respectively) with a 5x5x3 filter. We take the 5\*5\*3 filter and slide it over the complete image and along the way take the dot product between the filter and chunks of the input image.

Dot Product of Filter with single chunk of Input Image



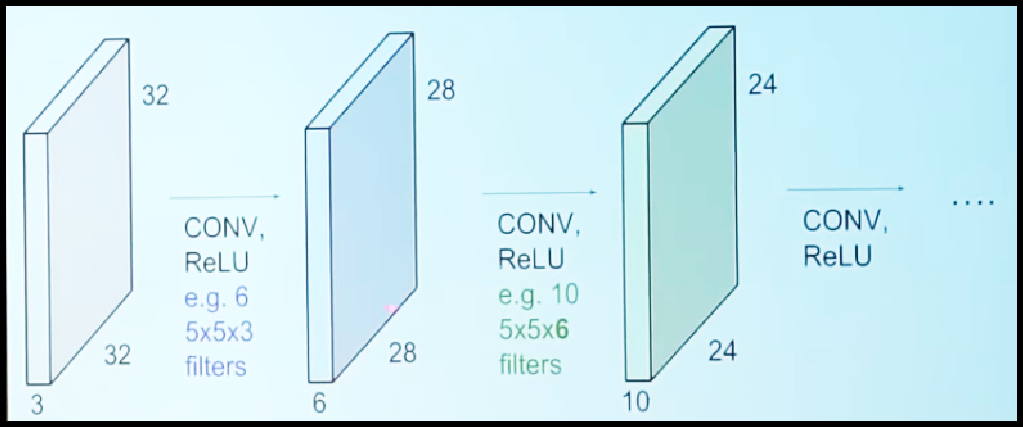
Dot Product or Convolve over all possible 5x5 spatial location in Input Image

The convolution layer is the main building block of a convolutional neural network. The convolution layer comprises of a set of independent filters (6 in the example shown). Each filter is independently convolved with the image and we end up with 6 feature maps of shape 28\*28\*1.



Input Image Convolving with a Convolutional layer of 6 independent filters

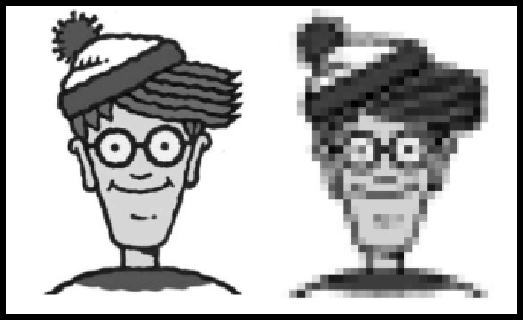
The CNN may consists of several Convolutional layers each of which can have similar or different number of independent filters. For example the following diagram shows the effect of two Convolutional layers having 6 and 10 filters respectively.



Input Image Convolving with two Convolutional layers having 6 and 10 filters respectively

All these filters are initialized randomly and become our parameters which will be learned by the network subsequently

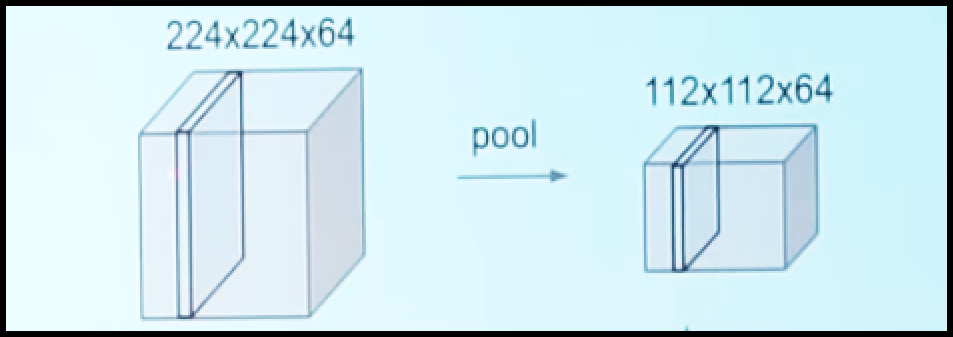
**Subsampling**

Inputs from the convolution layer can be “smoothened” to reduce the sensitivity of the filters to noise and variations. This smoothing process is called subsampling, and can be achieved by taking averages or taking the maximum over a sample of the signal. Examples of subsampling methods (for image signals) include reducing the size of the image, or reducing the color contrast across red, green, blue (RGB) channel.

Sub sampling Wally by 10 times. This creates a lower resolution image.

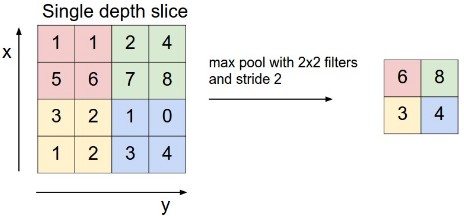
**Pooling**

A pooling layer is another building block of a CNN.



Pooling to reduce size from 224x224 to 112x112

Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently.

The most common approach used in pooling is max pooling in which maximum of a region taken as its representative. For example in the following diagram a 2x2 region is replaced by the maximum value in i.

Max Pooling

**Activation**

The activation layer controls how the signal flows from one layer to the next, emulating how neurons are fired in our brain. Output signals which are strongly associated with past references would activate more neurons, enabling signals to be propagated more efficiently for identification.

CNN is compatible with a wide variety of complex activation functions to model signal propagation, the most common function being the Rectified Linear Unit (ReLU), which is favored for its faster training speed.

**Fully Connected**

The last layers in the network are fully connected, meaning that neurons of preceding layers are connected to every neuron in subsequent layers. This mimics high level reasoning where all possible pathways from the input to output are considered.

**(During Training) Loss**

When training the neural network, there is additional layer called the loss layer. This layer provides feedback to the neural network on whether it identified inputs correctly, and if not, how far off its guesses were. This helps to guide the neural network to reinforce the right concepts as it trains. This is always the last layer during training.

**Implementation**

Algorithms used in training CNN are analogous to studying for exams using flash cards. First, you draw several flashcards and check if you have mastered theconcepts on each card. For cards with concepts that you already know discard them. For those cards with concepts that you are unsure of, put them back into the pile. Repeat this process until you are fairly certain that you know enough concepts to do well in the exam. This method allows you to focus on less familiar concepts by revisiting them often. Formally, these algorithms are called gradient descent algorithms for forward pass learning. Modern deep learning algorithm uses a variation called stochastic gradient descent, where instead of drawing the flashcards sequentially, you draw them at random. If similar topics are drawn in sequence, the learners might over­estimate how well they know the topic. The random approach helps to minimize any form of bias in the learning of topics.

Learning algorithms require feedback. This is done using a validation set where the CNN would make predictions and compare them with the true labels or ground truth. The predictions which errors are made are then fed backwards to the CNN to refine the weights learned, in a so-called backwards pass. Formally, this algorithm is called backpropagation of errors, and it requires functions in the CNN to be differentiable (almost).

CNNs are too complex to implement from scratch. Today, machine learning practitioners often utilize toolboxes developed such as Caffe, Torch, MatConvNet and Tensor flow for their work.

**Description of Overall Software Structure**

**Data Processing**

**Classifying Gesture**

**Training Modal**

**Classifying Gesture**

**Terminal**

**Camera**

**Text**

**Camera**

**Text**

**Recognise.py**

**Camera**

**Text**

**Camera**

**Text**

**Cam Text**

**Front End**

**Flask app**

**App.py**

**Training & Saving Modal**

**Cnn\_modal.py**

**Modal.h5**

**Camera feed**

**Capture.py**

**Dataset**

As shown in Figure, the project will be structured into 3 distinct functional blocks, Data Processing, Training, Classify Gesture. The block diagram is simplified in detail to abstract some of the minutiae:

• **Data Processing:**

The capture.py script contains functions to load the Raw Image Data and save the image data as numpy arrays into file storage. The capturre.py script will load the image data from data.npy and preprocess the image by resizing/rescaling the image, and applying filters and whitening to enhance features. During training the processed image data was split into training, and testing data and written to storage. Training also involves a load cnn.py script that loads the relevant data split into a Dataset class. For use of the trained model in classifying gestures.

• **Training:**

The training loop for the model is contained in cnn.py. The model is trained with hyperparameters obtained from a config file that lists the learning rate, batch size, image filtering, and number of epochs. The configuration used to train the model is saved along with the model architecture for future evaluation and tweaking for improved results. Within the training loop, the training and validation datasets are loaded as Dataloaders and the model is trained using Adam Optimizer . The model is evaluated every epoch on the validation set and the model with best validation accuracy is saved to storage for further evaluation and use. Upon finishing training, the training and validation error and loss is saved to the disk.

• **Classify Gesture:**

After a model has been trained, it can be used to classify a new ASL gesture that is available as a direct input from camera. The user inputs the gesture image and the recognize.py script will pass the image to process and load and preprocess the file the same way as the model has been trained. Model Generated through this process is saved as model.h5.

**Sources of Data**

**Data Collection**

The primary source of data for this project was the compiled dataset of American Sign Language (ASL) called the ASL Alphabet . The dataset is comprised of 52000 images which are 200x200 pixels. There are 26 total classes, each with 2000 images, 26 for the letters A-Z . The images taken from his laptop’s webcam. These photos were then cropped, rescaled, and labelled for use.

**  **

Letter A Letter B Letter C

**Data Pre-processing**

**Dilation, Opening, Closing And Erosion**

These are two fundamental image processing operations. These are used to removing noises, finding an intensity hole or bump in an image and many more.

**Cropping**

It is one of the most important and fundamental techniques in image processing, Cropping is used to get a particular part of an image. To crop an image. You just need the coordinates from an image according to your area of interest

**Scaling, Interpolations, And Re-Sizing**

Re-sizing is one of the easiest tasks in OpenCV. It provides a resize() function which takes parameters such as image, output size image, interpolation, x scale, and y scale



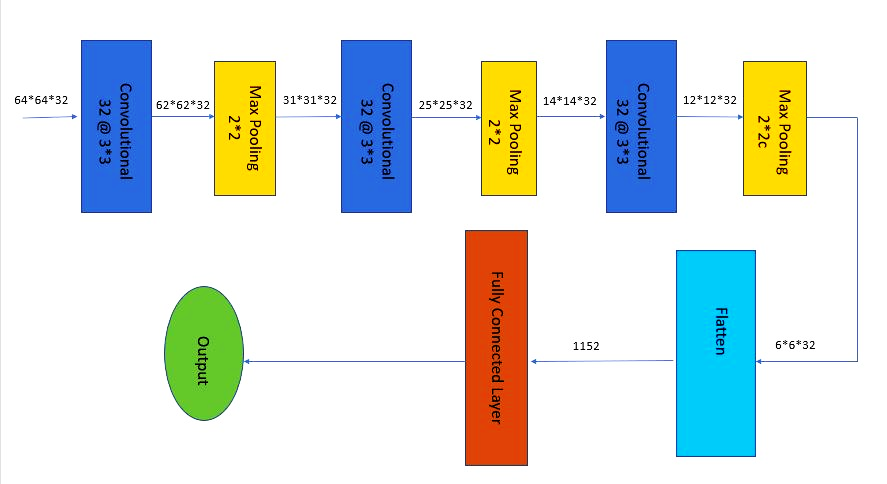
Image from Cam

Image After Contour

Image After Masking

**Machine Learning Model**

Overall Structure The model used in this classification task is a fairly basic implementation of a Convolutional Neural Network (CNN). As the project requires classification of images, a CNN is the go-to architecture. The basis for our model design came from Using Deep Convolutional Networks for Gesture Recognition in American Sign Language paper that accomplished a similar ASL Gesture Classification task . This model consisted of convolutional blocks containing two 2D Convolutional Layers with ReLU activation, followed by Max Pooling and Dropout layers. These convolutional blocks are repeated 3 times and followed by Fully Connected layers that eventually classify into the required categories. The kernel sizes are maintained at 3 X 3 throughout the model.



64\*64\*3

**Model Performance**

**Training and Validation**

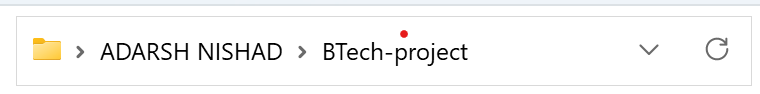
Our models were trained using Adam optimizer and Cross Entropy Loss. Adam optimizer is known for converging quickly in comparison with Stochastic Gradient Descent (SGD), even while using momentum. However, initially Adam would not decrease our loss thus we abandoned it to use SGD. Debugging Adam optimizer after our final presentation taught us that lowering learning rate significantly can help Adam to converge during training. Thus allowing us to train more models towards the end of our project.

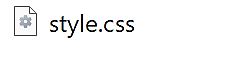
**Conclusion & Future Work**

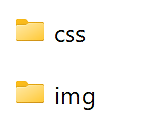
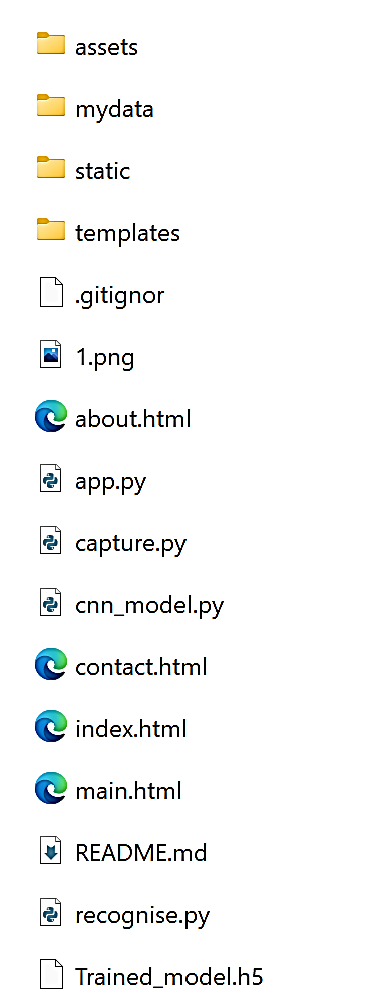
The technology improves the fields of artificial intelligence, machine learning and computer vision. They all together help us to bring the technology which helps society in a better way and improves the lives of human being. Many scientists and researcher use techniques like ANN, CNN etc to conduct research in the field of sign language recognition. many of them use high power of computing our research paper minimizes the high power of computer processing and make it available for every people. We classify 26 alphabet letters of American sign language successfully by using CNN.

We wish to extend our work further in recognising continuous sign language gestures with better accuracy. This method for individual gestures can also be extended for sentence level sign language. Also the current process uses two different models, training inception (CNN) . For future work one can focus on combining the two models into a single model.

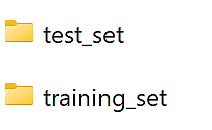
**Project Directory**

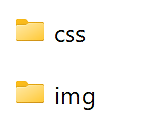
****

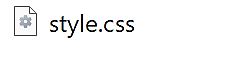
****

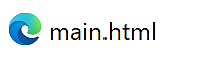
****

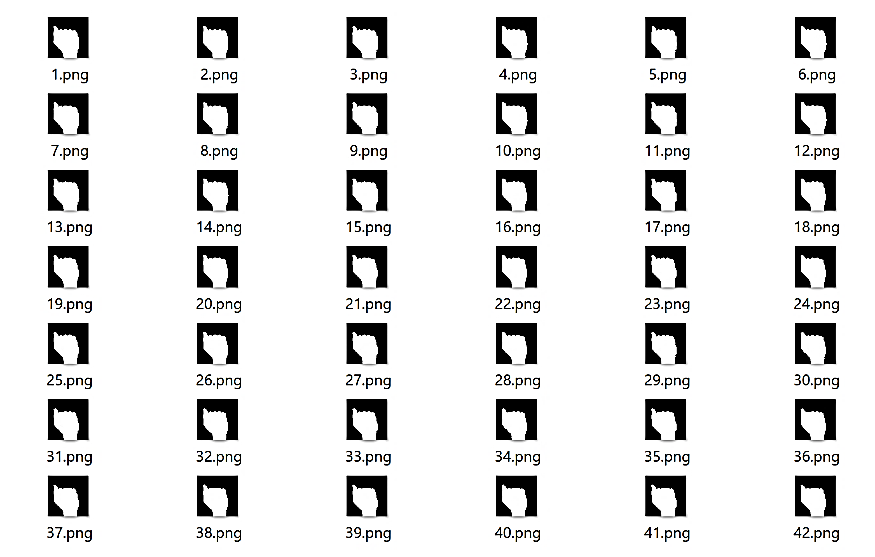
****

****

****

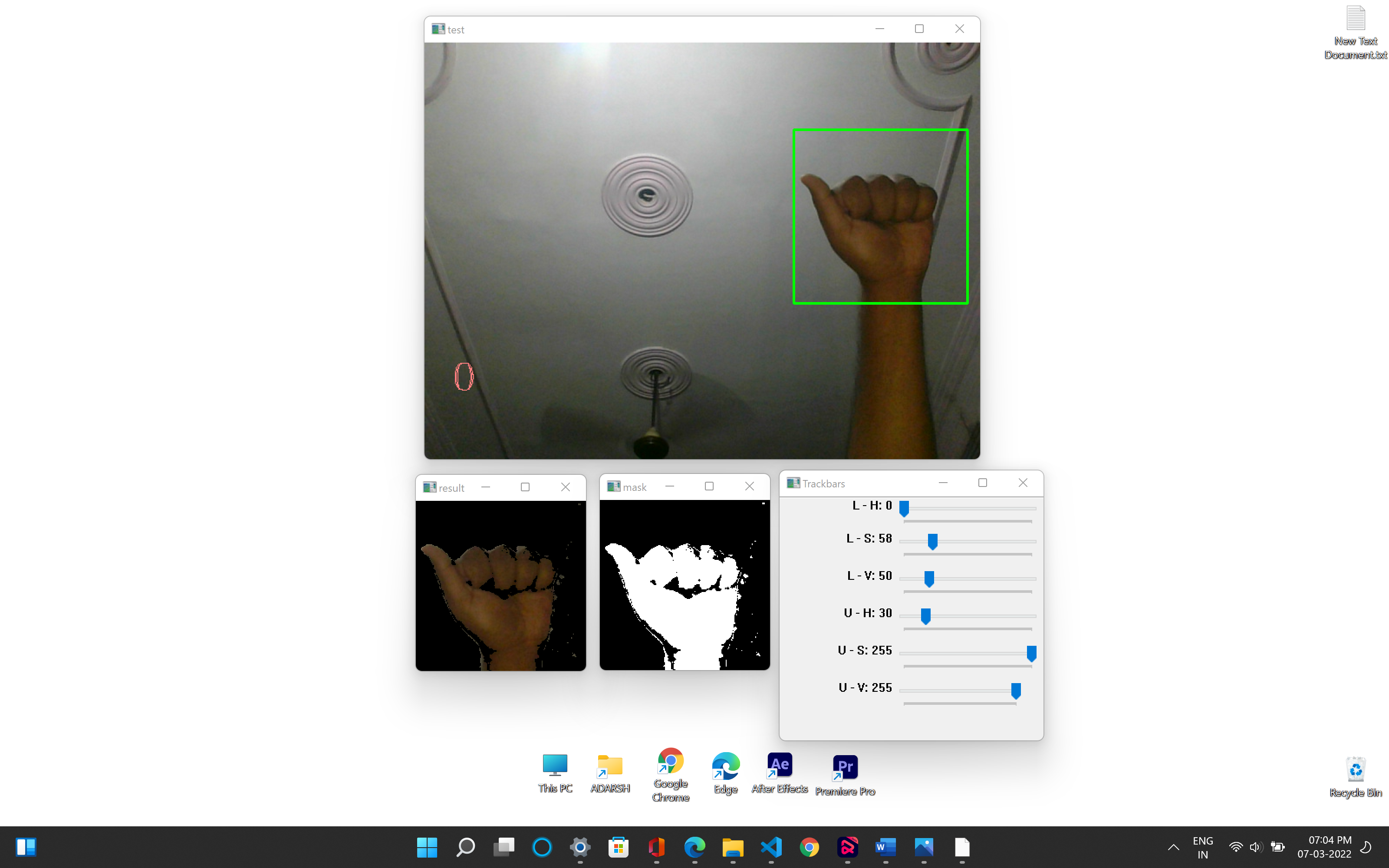
****

****

****

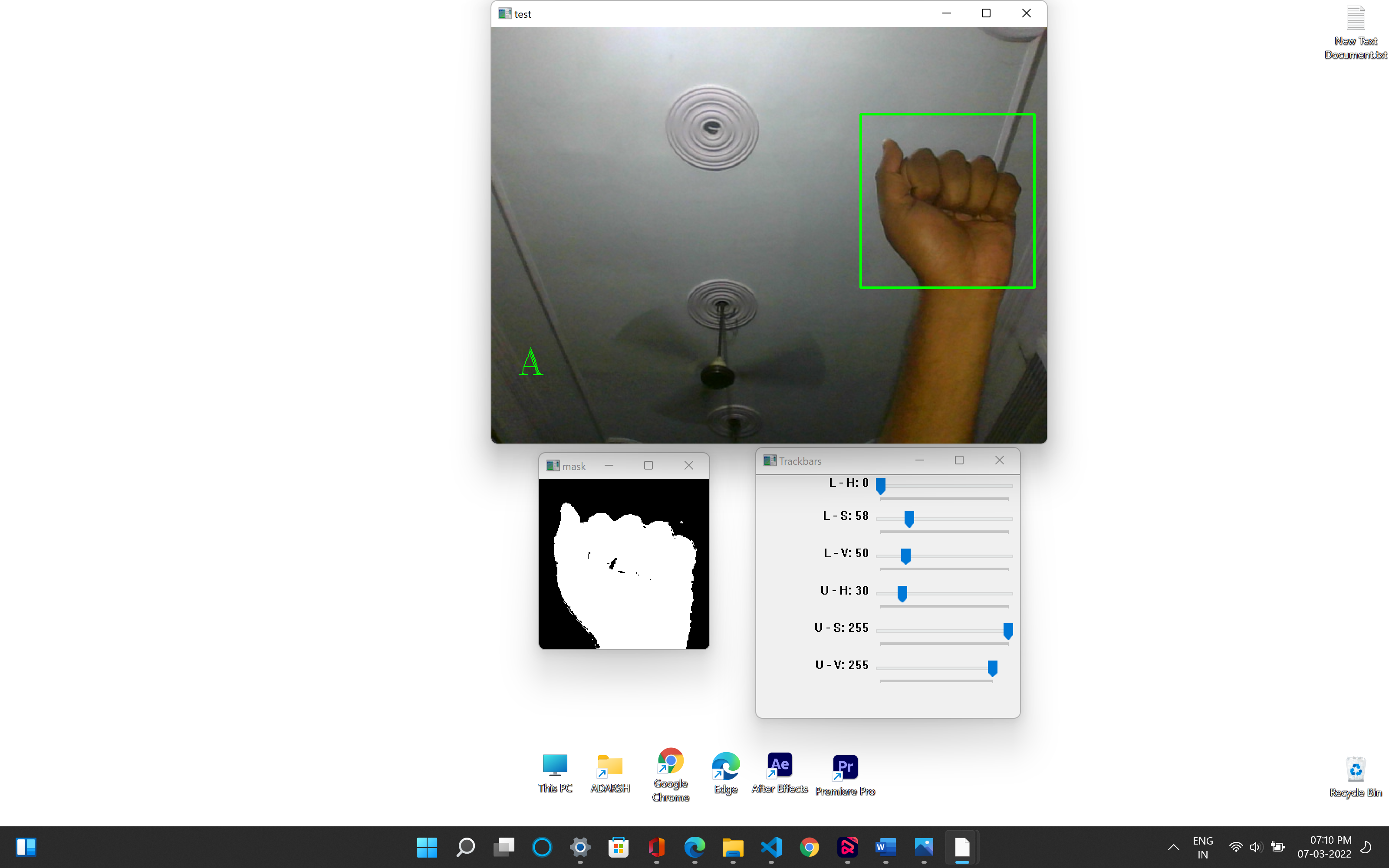
**Source Code**

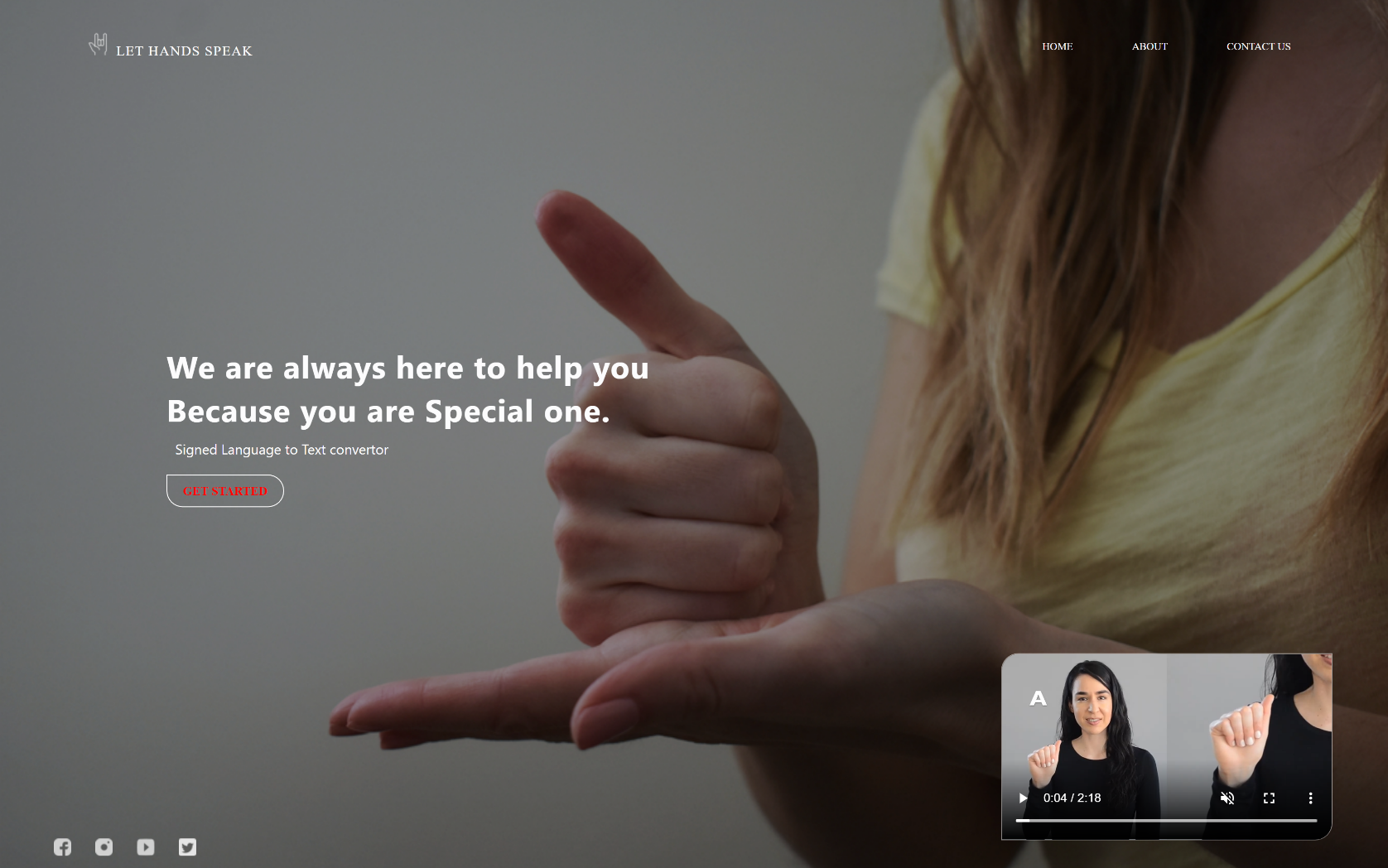
1. **Capture.py :** It captures dataset and stores them into respective folders.
2. import cv2
3. import time
4. import numpy as np
5. import os
6. def nothing(x):
7. pass
8. image\_x, image\_y = 64, 64
9. def create\_folder(folder\_name):
10. if not os.path.exists('./mydata/training\_set/' + folder\_name):
11. os.mkdir('./mydata/training\_set/' + folder\_name)
12. if not os.path.exists('./mydata/test\_set/' + folder\_name):
13. os.mkdir('./mydata/test\_set/' + folder\_name)
14. def capture\_images(ges\_name):
15. create\_folder(str(ges\_name))
16. cam = cv2.VideoCapture(0)
17. cv2.namedWindow("test")
18. img\_counter = 0
19. t\_counter = 1
20. training\_set\_image\_name = 1
21. test\_set\_image\_name = 1
22. listImage = [1, 2, 3, 4, 5]
23. cv2.namedWindow("Trackbars")
24. cv2.createTrackbar("L - H", "Trackbars", 0, 179, nothing)
25. cv2.createTrackbar("L - S", "Trackbars", 58, 255, nothing)
26. cv2.createTrackbar("L - V", "Trackbars", 50, 255, nothing)
27. cv2.createTrackbar("U - H", "Trackbars", 30, 179, nothing)
28. cv2.createTrackbar("U - S", "Trackbars", 255, 255, nothing)
29. cv2.createTrackbar("U - V", "Trackbars", 255, 255, nothing)
30. for loop in listImage:
31. while True:
32. ret, frame = cam.read()
33. frame = cv2.flip(frame, 1)
34. l\_h = cv2.getTrackbarPos("L - H", "Trackbars")
35. l\_s = cv2.getTrackbarPos("L - S", "Trackbars")
36. l\_v = cv2.getTrackbarPos("L - V", "Trackbars")
37. u\_h = cv2.getTrackbarPos("U - H", "Trackbars")
38. u\_s = cv2.getTrackbarPos("U - S", "Trackbars")
39. u\_v = cv2.getTrackbarPos("U - V", "Trackbars")
40. img = cv2.rectangle(frame, (425, 100), (625, 300),
41. (0, 255, 0), thickness=2, lineType=8, shift=0)
42. lower\_blue = np.array([l\_h, l\_s, l\_v])
43. upper\_blue = np.array([u\_h, u\_s, u\_v])
44. imcrop = img[102:298, 427:623]
45. hsv = cv2.cvtColor(imcrop, cv2.COLOR\_BGR2HSV)
46. mask = cv2.inRange(hsv, lower\_blue, upper\_blue)
47. result = cv2.bitwise\_and(imcrop, imcrop, mask=mask)
48. cv2.putText(frame, str(img\_counter), (30, 400),
49. cv2.FONT\_HERSHEY\_TRIPLEX, 1.5, (127, 127, 255))
50. cv2.imshow("test", frame)
51. cv2.imshow("mask", mask)
52. cv2.imshow("result", result)
53. if cv2.waitKey(1) == ord('c'):
54. if t\_counter <= 350:
55. img\_name = "./mydata/training\_set/" + \
56. str(ges\_name) + "/{}.png".format(training\_set\_image\_name)
57. save\_img = cv2.resize(mask, (image\_x, image\_y))
58. cv2.imwrite(img\_name, save\_img)
59. print("{} written!".format(img\_name))
60. training\_set\_image\_name += 1
61. if t\_counter > 350 and t\_counter <= 400:
62. img\_name = "./mydata/test\_set/" + \
63. str(ges\_name) + "/{}.png".format(test\_set\_image\_name)
64. save\_img = cv2.resize(mask, (image\_x, image\_y))
65. cv2.imwrite(img\_name, save\_img)
66. print("{} written!".format(img\_name))
67. test\_set\_image\_name += 1
68. if test\_set\_image\_name > 250:
69. break
70. t\_counter += 1
71. if t\_counter == 401:
72. t\_counter = 1
73. img\_counter += 1
74. elif cv2.waitKey(1) == 27:
75. break
76. if test\_set\_image\_name > 250:
77. break
78. cam.release()
79. cv2.destroyAllWindows()
80. ges\_name = input("Enter gesture name: ")
81. capture\_images(ges\_name)



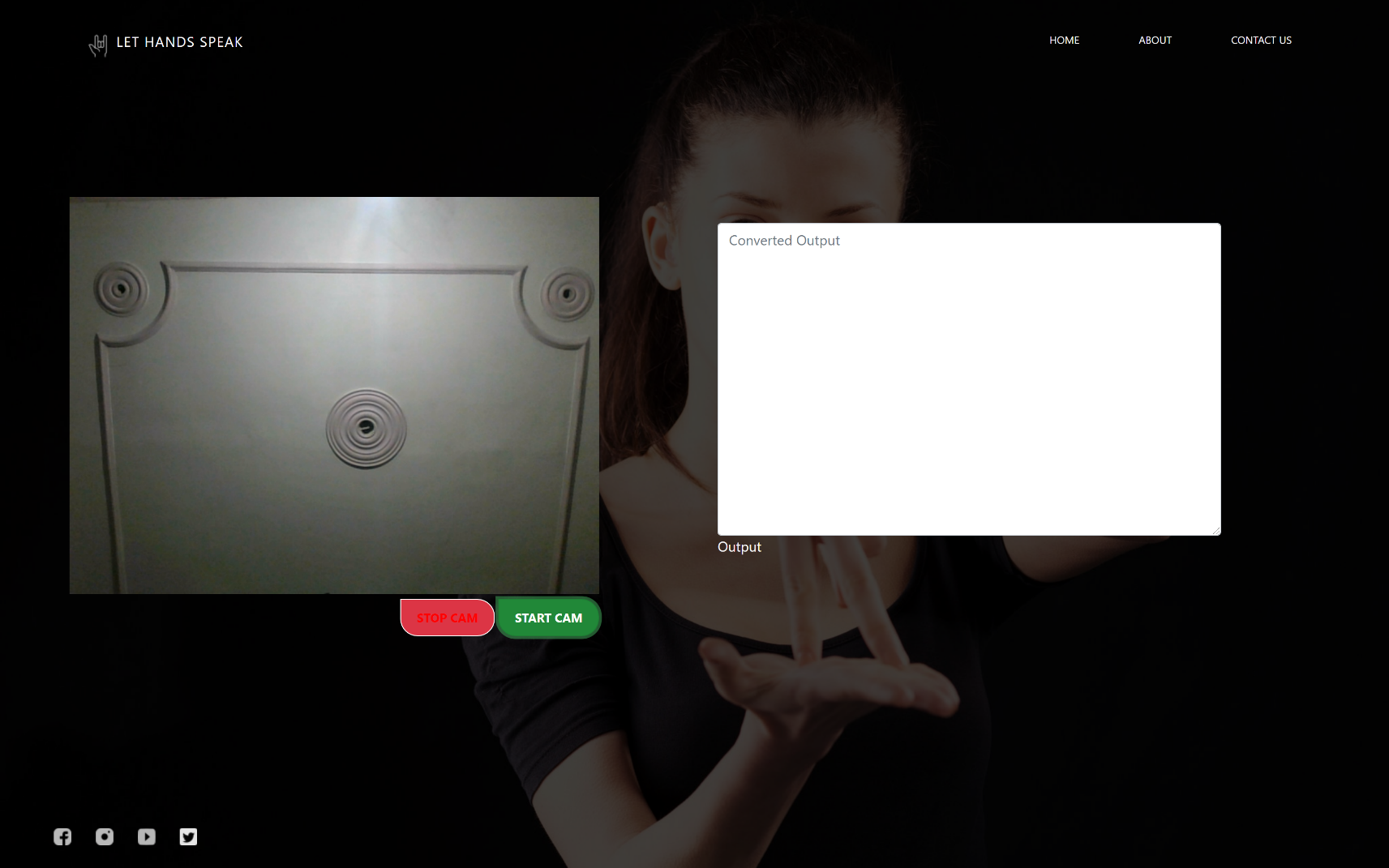
1. **Cnn\_model.py :** It is used to train ML modal from training dataset.
2. # importing the Keras libraries and packages
3. import matplotlib.pyplot as plt
4. import h5py
5. from keras.preprocessing.image import ImageDataGenerator
6. from keras.models import Sequential
7. from keras.layers import Convolution2D as Conv2D
8. from keras.layers import MaxPooling2D
9. from keras.layers import Flatten
10. from keras.layers import Dense, Dropout
11. from keras import optimizers
12. import os
13. os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'
14. # Initialing the CNN
15. classifier = Sequential()
16. # Adding first Convolutio Layer
17. classifier.add(Conv2D(32, (3,  3), input\_shape=(64, 64, 3), activation='relu'))
18. # Pooling
19. classifier.add(MaxPooling2D(pool\_size=(2, 2)))
20. # Adding second convolution layer
21. classifier.add(Conv2D(32,( 3,  3), activation='relu'))
22. classifier.add(MaxPooling2D(pool\_size=(2, 2)))
23. # Adding 3rd Concolution Layer
24. classifier.add(Conv2D(32, (3,  3), activation='relu'))
25. classifier.add(MaxPooling2D(pool\_size=(2, 2)))
26. # Step 3 - Flattening
27. classifier.add(Flatten())
28. # Step 4 - Full Connection
29. classifier.add(Dense(256, activation='relu'))
30. classifier.add(Dropout(0.5))
31. classifier.add(Dense(26, activation='softmax'))
32. # Compiling The CNN
33. classifier.compile(optimizer='adam',loss='categorical\_crossentropy',metrics=['accuracy'])
34. # Part 2 Fittting the CNN to the image
35. train\_datagen = ImageDataGenerator(rescale=1./255,shear\_range=0.2,zoom\_range=0.2,horizontal\_flip=True)
36. test\_datagen = ImageDataGenerator(rescale=1./255)
37. training\_set = train\_datagen.flow\_from\_directory('mydata/training\_set',target\_size=(64, 64),batch\_size=32,class\_mode='categorical')
38. test\_set = test\_datagen.flow\_from\_directory('mydata/test\_set',target\_size=(64, 64),batch\_size=32,class\_mode='categorical')
39. model = classifier.fit\_generator(training\_set,steps\_per\_epoch=800,epochs=25,validation\_data=test\_set,validation\_steps=6500)
40. # Saving the model
41. classifier.save('Trained\_model2.h5')
42. print(model.history.keys())
43. # summarize history for accuracy
44. plt.plot(model.history['acc'])
45. plt.plot(model.history['val\_acc'])
46. plt.title('model accuracy')
47. plt.ylabel('accuracy')
48. plt.xlabel('epoch')
49. plt.legend(['train', 'test'], loc='upper left')
50. plt.show()
51. # summarize history for loss
52. plt.plot(model.history['loss'])
53. plt.plot(model.history['val\_loss'])
54. plt.title('model loss')
55. plt.ylabel('loss')
56. plt.xlabel('epoch')
57. plt.legend(['train', 'test'], loc='upper left')
58. plt.show()
59. **App.py :** It is a flask script that connects ML modal to Front-end allow access to

camera.

1. # importing the Keras libraries and packages
2. from flask import Flask, render\_template, Response
3. import cv2
4. app = Flask(\_\_name\_\_)
5. camera = cv2.VideoCapture(0)
6. def generate\_frames():
7. while True:
8. # read the camera frame
9. success, frame = camera.read()
10. if not success:
11. break
12. else:
13. ret, buffer = cv2.imencode('.jpg', frame)
14. frame = buffer.tobytes()
15. yield(b'--frame\r\n'
16. b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
17. @app.route('/')
18. def index():
19. return render\_template('main.html')
20. @app.route('/video')
21. def video():
22. return Response(generate\_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')
23. if \_\_name\_\_ == "\_\_main\_\_":
24. app.run(debug=True)
25. **Recognise.py :** It predict the sign language through our trained ML modal
26. # importing the Keras libraries and packages
27. from keras.models import load\_model
28. import cv2
29. import numpy as np
30. def nothing(x):
31. pass
32. image\_x, image\_y = 64, 64
33. classifier = load\_model('Trained\_model1.h5')
34. def predictor():
35. import numpy as np
36. from keras.preprocessing import image
37. test\_image = image.load\_img('1.png', target\_size=(64, 64))
38. test\_image = image.img\_to\_array(test\_image)
39. test\_image = np.expand\_dims(test\_image, axis=0)
40. result = classifier.predict(test\_image)
41. if result[0][0] == 1:
42. return 'A'
43. elif result[0][1] == 1:
44. return 'B'
45. elif result[0][2] == 1:
46. return 'C'
47. elif result[0][3] == 1:
48. return 'D'
49. elif result[0][4] == 1:
50. return 'E'
51. elif result[0][5] == 1:
52. return 'F'
53. elif result[0][6] == 1:
54. return 'G'
55. elif result[0][7] == 1:
56. return 'H'
57. elif result[0][8] == 1:
58. return 'I'
59. elif result[0][9] == 1:
60. return 'J'
61. elif result[0][10] == 1:
62. return 'K'
63. elif result[0][11] == 1:
64. return 'L'
65. elif result[0][12] == 1:
66. return 'M'
67. elif result[0][13] == 1:
68. return 'N'
69. elif result[0][14] == 1:
70. return 'O'
71. elif result[0][15] == 1:
72. return 'P'
73. elif result[0][16] == 1:
74. return 'Q'
75. elif result[0][17] == 1:
76. return 'R'
77. elif result[0][18] == 1:
78. return 'S'
79. elif result[0][19] == 1:
80. return 'T'
81. elif result[0][20] == 1:
82. return 'U'
83. elif result[0][21] == 1:
84. return 'V'
85. elif result[0][22] == 1:
86. return 'W'
87. elif result[0][23] == 1:
88. return 'X'
89. elif result[0][24] == 1:
90. return 'Y'
91. elif result[0][25] == 1:
92. return 'Z'
93. cam = cv2.VideoCapture(0)
94. cv2.namedWindow("Trackbars")
95. cv2.createTrackbar("L - H", "Trackbars", 0, 179, nothing)
96. cv2.createTrackbar("L - S", "Trackbars", 58, 255, nothing)
97. cv2.createTrackbar("L - V", "Trackbars", 50, 255, nothing)
98. cv2.createTrackbar("U - H", "Trackbars", 30, 179, nothing)
99. cv2.createTrackbar("U - S", "Trackbars", 255, 255, nothing)
100. cv2.createTrackbar("U - V", "Trackbars", 255, 255, nothing)
101. cv2.namedWindow("test")
102. img\_counter = 0
103. img\_text = ''
104. while True:
105. ret, frame = cam.read()
106. frame = cv2.flip(frame, 1)
107. l\_h = cv2.getTrackbarPos("L - H", "Trackbars")
108. l\_s = cv2.getTrackbarPos("L - S", "Trackbars")
109. l\_v = cv2.getTrackbarPos("L - V", "Trackbars")
110. u\_h = cv2.getTrackbarPos("U - H", "Trackbars")
111. u\_s = cv2.getTrackbarPos("U - S", "Trackbars")
112. u\_v = cv2.getTrackbarPos("U - V", "Trackbars")
113. img = cv2.rectangle(frame, (425, 100), (625, 300),
114. (0, 255, 0), thickness=2, lineType=8, shift=0)
115. lower\_blue = np.array([l\_h, l\_s, l\_v])
116. upper\_blue = np.array([u\_h, u\_s, u\_v])
117. imcrop = img[102:298, 427:623]
118. hsv = cv2.cvtColor(imcrop, cv2.COLOR\_BGR2HSV)
119. mask = cv2.inRange(hsv, lower\_blue, upper\_blue)
120. cv2.putText(frame, img\_text, (30, 400),
121. cv2.FONT\_HERSHEY\_TRIPLEX, 1.5, (0, 255, 0))
122. cv2.imshow("test", frame)
123. cv2.imshow("mask", mask)
124. # if cv2.waitKey(1) == ord('c'):
125. img\_name = "1.png"
126. save\_img = cv2.resize(mask, (image\_x, image\_y))
127. cv2.imwrite(img\_name, save\_img)
128. print("{} written!".format(img\_text))
129. img\_text = predictor()
130. ****    if cv2.waitKey(1) == 27:
131. break
132. cam.release()
133. cv2.destroyAllWindows()
134. **Index.html :** This is Home page of our website.
135. <!doctype html>
136. <html>
137. <head>
138. <meta charset="utf-8">
139. <title>Home | Let Hands Speak </title>
140. <link href="assets/css/style.css" rel="stylesheet" type="text/css">
141. </head>
142. <body>
143. <div class="slider">
144. <div class="content">
145. <!-- Navigation Bar-->
146. <div class="navbar">
147. <a href="index.html" class="logo-text"><img src="assets/img/logo.png" alt="logo" class="logo">Let Hands Speak</a>
148. <ul>
149. <li><a href="index.html">Home</a></li>
150. <li><a href="about.html">About</a></li>
151. <li><a href="contact.html">Contact us</a></li>
152. </ul>
153. </div>
154. <!-- Conatent -->
155. <div class="containers">
156. <h1>We are always here to help you <br>Because you are Special one.</h1>
157. <p>Signed Language to Text convertor </p><br>
158. <a href="main.html" target="\_blank">Get Started</a>
159. </div>
160. <!-- Footer-->
161. <footer>
162. <ul>
163. <li><a href="" target="\_blank"><img src="assets/img/icon-fb.png" alt="logo"></a></li>
164. <li><a href="" target="\_blank"><img src="assets/img/icon-insta.png" alt="logo"></a></li>
165. <li><a href="" target="\_blank"><img src="assets/img/icon-yt.png" alt="logo"></a></li>
166. <li><a href="" target="\_blank"><img src="assets/img/icon-tweet.png" alt="logo"></a></li>
167. </ul>
168. </footer>
169. <div class="video-box">
170. <video controls autoplay muted width="380">
171. <source src="assets/img/video1.mp4"
172. type="video/mp4">
173. </video>
174. </div>
176. </div>
177. </div>
178. </body>
179. </html>

****

1. **Main.html :** This page converts sign language into text.
2. <!DOCTYPE html>
3. <html>
4. <head>
5. <meta charset="utf-8" />
6. <title>Recognition | Let Hands Speak</title>
7. <link
8. href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css"
9. rel="stylesheet"
10. />
11. <script
12. src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"
13. type="text/javascript"
14. ></script>
15. <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.bundle.min.js"></script>
16. <link
17. rel="stylesheet"
18. href="{{ url\_for('static', filename='css/style.css') }}"
19. type="text/css"
20. />
21. </head>
22. <body>
23. <div class="bg">
24. <div class="layer">
25. <!-- Navigation Bar-->
26. <div class="navbar">
27. <a href="index.html" class="logo-text"
28. ><img
29. src="{{ url\_for('static', filename='img/logo.png') }}"
30. alt="logo"
31. class="logo"
32. />Let Hands Speak</a
33. >
34. <ul>
35. <li><a href="index.html">Home</a></li>
36. <li><a href="about.html">About</a></li>
37. <li><a href="contact.html">Contact us</a></li>
38. </ul>
39. </div>
40. <!-- Conatent -->
41. <div class="container" style="left: 5%">
42. <div class="row">
43. <div class="col-sm-6">
44. <div class="booth">
45. <!--video id="video" width="100%"
46. height="100%" autoplay>
47. </video-->
48. <img src="{{ url\_for('video') }}" width="100%" height="100%" />
49. </div>
50. <div class="text-right">
51. <a href="#!" class="btn btn-danger" onClick="stop()">
52. Stop Cam
53. </a>
54. <a href="#!" class="btn btn-success" onClick="start()">
55. Start Cam
56. </a>
57. </div>
58. </div>
59. <div class="offset-1 col-sm-5">
60. <div class="form-floating" style="margin-top: 30px">
61. <textarea
62. class="form-control"
63. placeholder="Converted Output"
64. id="floatingTextarea2"
65. style="height: 360px; width: 580px"
66. ></textarea>
67. <label for="floatingTextarea2">Output</label>
68. </div>
69. </div>
70. </div>
71. </div>
72. <!-- Footer-->
74. <footer>
75. <ul>
76. <li>
77. <a href="" target="\_blank"
78. ><img src="{{ url\_for('static', filename='img/icon-fb') }}" alt="logo"
79. /></a>
80. </li>
81. <li>
82. <a href="" target="\_blank"
83. ><img src="{{ url\_for('static', filename='img/icon-insta') }}" alt="logo"
84. /></a>
85. </li>
86. <li>
87. <a href="" target="\_blank"
88. ><img src="{{ url\_for('static', filename='img/icon-yt') }}" alt="logo"
89. /></a>
90. </li>
91. <li>
92. <a href="" target="\_blank"
93. ><img src="{{ url\_for('static', filename='img/icon-tweet') }}" alt="logo"
94. /></a>
95. </li>
96. </ul>
97. </footer>
98. </div>
99. </div>
100. <script>
101. var stop = function () {
102. var stream = video.srcObject;
103. var tracks = stream.getTracks();
104. for (var i = 0; i < tracks.length; i++) {
105. var track = tracks[i];
106. track.stop();
107. }
108. video.srcObject = null;
109. };
110. var start = function () {
111. var video = document.getElementById('video'),
112. vendorUrl = window.URL || window.webkitURL;
113. if (navigator.mediaDevices.getUserMedia) {
114. navigator.mediaDevices
115. .getUserMedia({ video: true })
116. .then(function (stream) {
117. video.srcObject = stream;
118. })
119. .catch(function (error) {
120. console.log('Something went wrong!');
121. });
122. }
123. };
124. $(function () {
125. start();
126. });
127. </script>
128. </body>
129. </html>



**References**

[1] T. Bohra, S. Sompura, K. Parekh and P. Raut, "Real-Time Two-Way Communication System for Speech and Hearing-Impaired Using Computer Vision and Deep Learning," 2019 International Conference on Smart Systems and Inventive Technology (ICSSIT), Tirunelveli, India, 2019, pp. 734-739, doi: 10.1109/ICSSIT46314.2019.8987908.

[2] Singha, Joyeeta & Das, Karen. (2013), “Recognition of Indian Sign Language in Live Video,” International Journal of Computer Applications. 70. 10.5120/12174-7306.

[3] H. Muthu Mariappan and V. Gomathi, "Real-Time Recognition of Indian Sign Language," 2019 International Conference on Computational Intelligence in Data Science (ICCIDS), Chennai, India, 2019, pp. 1-6, doi: 10.1109/ICCIDS.2019.8862125.

[4] K. Bantupalli and Y. Xie, "American Sign Language Recognition using Deep Learning and Computer Vision," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 4896-4899, doi: 10.1109/BigData.2018.8622141.

[5] S. Kim, Y. Ji and K. Lee, "An Effective Sign Language Learning with Object Detection Based ROI Segmentation," 2018 Second IEEE International Conference on Robotic Computing (IRC), 2018, pp. 330-333, doi: 10.1109/IRC.2018.00069.

[6] Deaf Cambridge Dictionary (2018). Retrieved from Cambridge Dictionary: <https://dictionary.cambridge.org/dictionary/english/deaf>

[7] Mehreen Hurroo , Mohammad Elham, 2020, Sign Language Recognition System using Convolutional Neural Network and Computer Vision, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 09, Issue 12 (December 2020),