Real Time Sign Language Recognition

Gargi Lohia (19BCB0049)

Sona Ann Joseph (19BCE2262)

Abin Philip Varghese (20BCE2810)

CSE4019 Image Processing

J Component Project

Faculty: Prof. Sureshkumar N

I. Abstract

Sign language is a visual communication technique that uses hand gestures, facial expressions and body language to convey messages and emotions. Signing is primarily used by the deaf and also those who have a disability or condition that makes them unable to physically speak. Contrary to popular belief, sign language isn't universal. There exist different sign languages too.

This project aims to address issues that tend to isolate people with disabilities. A model that takes hand gestures of sign language as input and returns the corresponding character as output has a wide range of applications. It can help inculcate a sense of belonging and helps the deaf and hard of hearing to converse with people who don't sign.

II. Keywords

ISL, Gestures, CNN, ROI, Segmentation, Contours, Thresholding

III. Abbreviations

ISL	Indian Sign Language
CNN	Convolution Neural Network
ROI	Region Of Interest

IV. Introduction

Sign Language is the mode of communication for the deaf and hard of hearing community. Very few people understand sign language and are able to have conversation with these communities. It has been established that there exists a communication gap between disabled people, due to this same reason. Through this project we aim to build a model that can help resolve some of these issues. CNNs are known to be able to automate the process of feature construction. So, this implementation follows the creation of a sign language recognition model that will be able to generalize users and surroundings not occurring during training, i.e., it will take live input and an thus help the deaf community to have active conversation with others regardless of the language barrier. The same application can also work as a sign to text editor.

Though the deaf have their own community and cultures, being able to communicate, or at least understand ISL (or corresponding sign language of the region) will immensely reduce the social gap that people tend to create with individuals that are deaf or hard of hearing. It is difficult to communicate with the written word as it is way slower, and people who aren't used to communicating with the spoken word are generally less able to communicate well in the tongue spoken by others. So a model that can simplify this process is of high relevance. This project uses various image processing techniques to create a new dataset of ISL gestures and uses this dataset to train the CNN model. The use of contouring and segmentation helps to create a more accurate and reliable dataset, leading to a more accurate prediction model. The conversion of the captured RGB image into thresholded binary images makes it easier to build the model. This also makes it easier to extract the edges (or contours) or the hand gesture to create the dataset and also during the live gesture recognition.

In conclusion, a model that can recognize sign language gestures will help in efficient communication and also create a more inclusive environment for the deaf community.

V. Literature Survey

Serial No.	1.	
	Paper Name	Deep Convolution Neural Networks for Image Classification: A Comprehensive Review
	Authors	Waseem Rawat Zenghui Wang
	Year	2017
	Methodology	This paper discusses a comprehensive review of CNN's for image classification tasks by categorizing progression based on early development, contribution to deep learning and rapid advancements over the years. DCNNs dominate numerous image classification related challenges and competitions, even to a level where they have surpassed human level performance. Various aspects of DCNN computation can be carried out using techniques established in digital image processing (FFTs (Fast Fourier Transform), DFTs (Discrete Fourier Transform), DCTs (Discrete Cosine Transform) and Convolution Theorem.
	Future Work	In recent years, DCNNs have been scrutinized for their classification performance, robustness and computational characteristics, leading to the discovery of many challenges to address. There are various open issues and trends to further analyze such as classification accuracy of DCNNs and the lack of theoretical proof of their success.

Serial No.	2.	
	Paper Name	Sign Language Recognition Application Systems for Deaf-Mute People: A Review Based on Input-Process-Output
	Authors	Suharjito, Ricky Anderson, Fanny Wiryana, Meita Chandra Ariesta, Gede Putra Kusuma
	Year	2017
	Methodology	Comparative study of the various processing methods: · Hidden Markov Model - combines various algorithms to perform Hand Shape Classification, Orientation Classification, and Movement Classification. · Convolution Neural Networks - 2D CNN for speech and image recognition and implements 3D CNN by adding a motion feature. · Artificial Neural Networks – uses ANN to recognize the feature described in a 2D curve using Elliptical Fourier Descriptor. · Self-Organizing Map – Kohonen Networks recognize images based on minimum Euclidean distance for similarity measurements. · Support Vector Machine – it classifies feature vectors of the extracted features, such as location, angle, shape invariant moments, fingertips, and trajectory using spatial domain. · Wavelet family method – detects peak of signals, matches signals of gesture with those in gesture database. · SIFT algorithm – compares extracted features with images stored in the database.
	Future Work	Most research work in this field is done using small and self made vocabularies. A large database is not available for most countries involved in the development of Sign Language Recognition. Applications of the Kinect-based data (provides color and depth stream videos) is also limited.

Serial Number	3	
	Paper Name	American Sign Language Recognition System: An Optimal Approach
	Authors	Shivashankara S Srinath S
	Year	2018
	Methodology	Identification of ASL alphanumeric characters is done mainly depending on hand and fingers only. The various components involved are: · Processing the input image · Computation of the region properties of the preprocessed image · Transliteration from treated image to text This method uses YCbCr color space (instead of HSV color model) to optimize skin color clustering. A comparative recognition rate analysis shows that the proposed method yields a better rate of 93.05% compared to existing ASL recognition techniques.
	Future Work	Two among the 26 ASL alphabet gestures require hand movements, hence they would require video frames to be captured and processed. A future addition could be the recognition of rotation and distance invariant ASL gestures and also can be extended to gestures in different backgrounds, locations or lighting. Further it can be modified to recognize complete words and sentences which require video processing.

Serial No.	4.	
	Paper Name	Indian Sign Language Recognition System
	Authors	Yogeshwar I Rokade Prashant M Jadav
	Year	2017
	Methodology	Two different approaches are mentioned: · Glove based approach – the signer wears a colored or sensor glove to simplify the segmentation phase, but this also poses a limitation as the glove is mandatory. · Vision based approach – uses algorithms to process, detect and track hand and facial expression. Further implementation of the vision based approach is done which consists of different phases like Image Acquisition, Hand Object Detection (Segmentation, Filter and Noise Removal, Feature Extraction) and Classification (Train and Test phase) done using ANN and SVM. Both models give accuracy rates between 90% – 95% with a fixed number of features.
	Future Work	By performing further reshuffling and refitting, changing the number of features, we can identify the optimal conditions for each model to give the best accuracy rates.

Serial No.	5.	
	Paper Name	Indian Sign Language Character Recognition
	Authors	Sanil Jain K V Sameer Raja
	Year	
	Methodology	Consists of the following phases: · Image segmentation – segment skin part from image and consider remaining part as noise, uses HSV model to separate color and intensity components. · Feature extraction – Scale Inverse Feature Transform (SIFT) to compute key points in the image, which is more accurate than manually describing features. · Machine learning on feature vectors – SVM, Random Forest, and Hierarchical Classification.
	Future Work	The results obtained were as follows: · SVM (54.63% accuracy) · Random Forest (46.45% accuracy) · Hierarchical Classification (53.23% accuracy) Using higher quality datasets with more variations can boost the accuracy. More complex models, like ANN or deep learning methods can improve results.

Serial No.	6.	
	Paper Name	Automatic Indian Sign Language Recognition for Continuous Video Sequence
	Authors	Joyeeta Singha, Karen Das
	Year	2015
	Methodology	Consists of four major phases- Data Acquisition, Pre-processing, Feature Extraction and Classification. Pre-processing phase includes Skin Filtering and histogram matching. Eigenvectors were extracted as features and Eigen value weighted Euclidean distance based classification was used.
	Future Work	This implementation goes beyond just still image recognition, to video sequence recognition of alphabets in ISL. It can further be modified to predict even words and sentences.

VI. Problem Statement

To design a real time software system that will be able to recognize Sign Language hand gestures using image processing and deep learning techniques.

VII. Architecture

Modules Description

1. Capturing the sign language gestures of alphabets for dataset creation:

   For the purpose of demonstration, we will create the test and train sets with the first 10 alphabets. It can easily be extended to all alphabets and numbers. The major image processing functions occur in this module

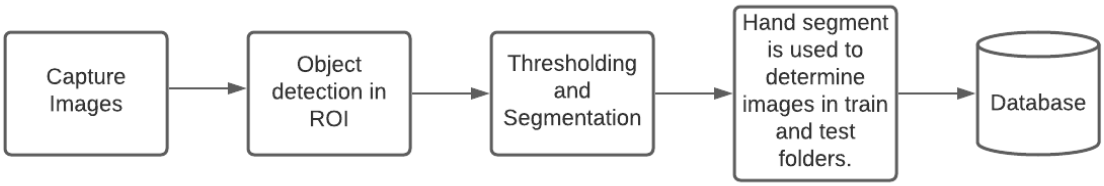
   A. Capturing live feed for train and test sets:
   - This will involve capturing a number of images that have an object (the hand in this case) in the ROI.
   - The live field is obtained with the help of OpenCV.
   - The ROI is a box that is our region of interest, inside which the hand will be detected.
   - These images are stored in two folders: test and train.
   - Each of the folders has subfolders, where each subfolder contains multiple images of the same alphabet.

   B. ROI background and foreground differentiation:
   - This will be done using accumulated weights.
   - The accumulated weighted average of the background is calculated.
   - This average is subtracted from the weighted average of every captured frame to locate objects.

   C. Segmentation (threshold, contouring)
   - The threshold for every frame will be calculated.
   - The outermost contours for each object will be returned.
   - When the contours are detected, the images will be saved in the test and train folders.

D. Putting it all together:

- The program will capture 300 images for the train set and 40 images for the test set, for each specified alphabet/number.
- The frame is flipped to prevent capture of an inverted image.
- Once a hand is detected in the ROI, the functions defined to calculate accumulated weighted average and to perform segmentation will be called to draw contours around the hand segment to determine the thresholded image which will be saved to test and train folders.



2. Training a CNN on this dataset:

Loading the captured images and designing a CNN. The model will be trained and tested by comparing two optimization algorithms - SGD (Stochastic Gradient Descent) and Adam (Adagrad + RMSProp) to find which gives better accuracy.

A. Load the created gesture images

- 300 train images and 40 test images for each alphabets is loaded from the respective folders
- They are split into train and test batches

B. Design the CNN model

- Build the model and compare the accuracy and loss of both the optimization algorithms to find the optimal model
- Fit the model on the train batch and test batch for 10 epochs
- Calculate accuracy and loss of the final model
- Create a dictionary to map the images to appropriate labels

- Test the model on a small set of test data by predicting labels and comparing them with the actual labels.

3. Program to predict the gesture:

Here we follow the same steps used to capture the dataset (detecting background, segmenting the foreground from background) in order to detect the required hand gesture in the live ROI. This becomes the new input for the saved detection model, which then predicts the accurately translated alphabet.

A. Load the model

- The fitted and saved final model is loaded

B. Define functions

- As in the case of dataset creation, here also we need to calculate accumulated weight, differentiate foreground and background; perform segmentation by contouring and thresholding.

C. Capture live input

- Getting a new input for the model to predict its value
- The captured image is processed by the functions and the thresholded hand image is input to the final model

D. Prediction Result

- The model predicts the alphabet for the input gesture

VIII. Implementation and Result

1. Dataset Creation:

```
import numpy as np
import cv2

cam = cv2.VideoCapture(0, cv2.CAP_DSHOW)

nf = 0
ele = 'A'
num_imgs = 0

bg = None
acc_weight = 0.5

def accumulated_average(frame, accumulated_weight):
    global bg
    if bg is None:
        bg = frame.copy().astype("float")
        return None
    cv2.accumulateWeighted(frame, bg, accumulated_weight)
```

Code Snippet 1.1 Function to calculate accumulated weighted average

```
def segmentation(frame, threshold=25):
    global bg
    diff = cv2.absdiff(bg.astype("uint8"), frame)
    _ , thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY)
    contours, hierarchy = cv2.findContours(thresholded.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if len(contours) == 0:
        return None
    else:
        segmented_hand_max_cont = max(contours, key=cv2.contourArea)
        return (thresholded, segmented_hand_max_cont)


rt = 200
rb = 400
rr = 200
rl = 400

while True:
    ret, frame = cam.read()

    frame = cv2.flip(frame,1)  #frame is flipped because it is captured as a mirror image
    f_copy = frame.copy()  #creating a copy of the frame
    roi = frame[rt:rb, rr:rl]

    # processing ROI part of the image
    gray = cv2.cvtColor(roi, cv2.COLORBGR2GRAY)
```

Code Snippet 1.2. Function to perform image segmentation. Defining ROI dimensions.

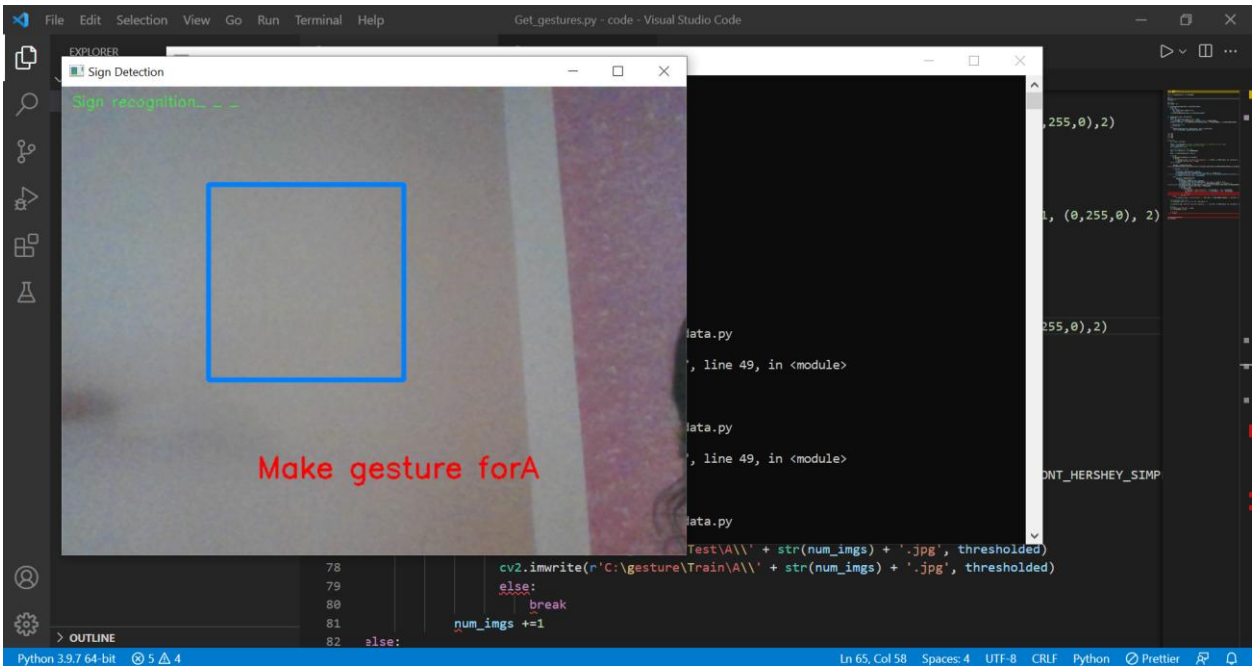
```
while True:
    ret, frame = cam.read()

    frame = cv2.flip(frame,1)  #frame is flipped because it is captured as a mirror image
    f_copy = frame.copy()  #creating a copy of the frame
    roi = frame[rt:rb, rr:rl]

    # processing ROI part of the image
    gray = cv2.cvtColor(roi, cv2.COLORBGR2GRAY)

    gray = cv2.GaussianBlur(gray, (9,9),0)

    if nf< 60:
        accumulated_average(gray, acc_weight)
        if nf<=59:
            cv2.putText(f_copy, "Collecting the background...", (70,300), cv2.FONT_ITALIC, 0.8, (0,255,0),
            cv2.imshow("Sign Detection", f_copy)
    elif nf <= 300:

        img_hand = segmentation(gray)
        cv2.putText(f_copy, "Make gesture for " + str(ele), (200, 400), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,25

        if img_hand is not None:

            thresholded, segmented_hand = img_hand
            cv2.drawContours(f_copy, [segmented_hand + (rr,rt)], -1, (255,0,0),1)
            cv2.putText(f_copy, str(nf) + "For" + str(ele), (70,45),cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),2
            cv2.imshow("Thresholded image", thresholded)
        else:
             img_hand = segmentation(gray)
```

Get_gestures.py ● Train_CNN.py ● model_final.py ●

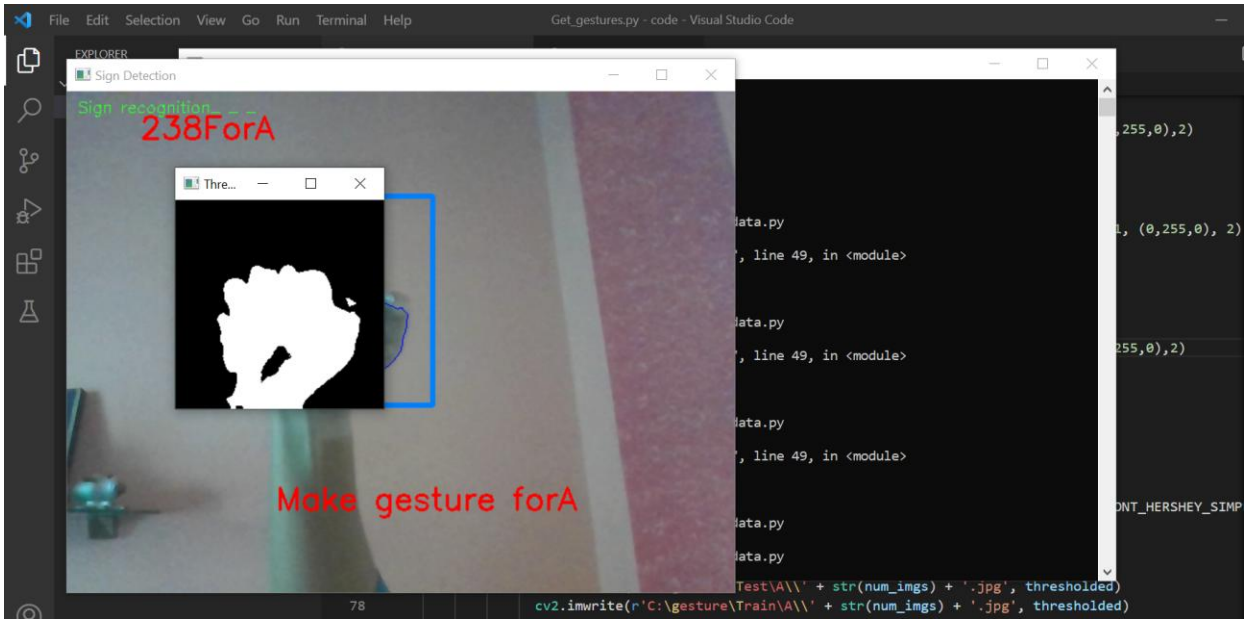
C: > Users > dell > Desktop > SLR > Get_gestures.py

```
68              img_hand = segmentation(gray)
69              if img_hand is not None:
70                  thresholded, segmented_hand = img_hand
71                  cv2.drawContours(f_copy, [segmented_hand + (rr, rt)], -1, (255, 0, 0),1)
72                  cv2.putText(f_copy, str(nf), (70, 45), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)
73                  cv2.putText(f_copy, str(num_imgs) + 'images' +"For" + str(ele), (200, 400), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)
74                  cv2.imshow("Thresholded Hand Image", thresholded)
75                  if num_imgs <= 300:
76                      if num_imgs<=39:
77                          cv2.imwrite(r'C:\gesture\Test\A\\' + str(num_imgs) + '.jpg', thresholded)
78                          cv2.imwrite(r'C:\gesture\Train\A\\' + str(num_imgs) + '.jpg', thresholded)
79                      else:
80                         break
81                      num_imgs +=1
82                  else:
83                      cv2.putText(f_copy, 'No hand detected...', (200, 400), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)
84
85      # Drawing ROI on frame copy
86      cv2.rectangle(f_copy, (rl, rt), (rr, rb), (255,128,0), 3)
87
88      cv2.putText(f_copy, "Hand sign recognition_ _ _", (10, 20), cv2.FONT_ITALIC, 0.5, (51,255,51), 1)
89
90      nf += 1
91      cv2.imshow("Sign Detection", f_copy)
92      k = cv2.waitKey(1) & 0xFF
93
94      if k == 27:
95          break
96
```

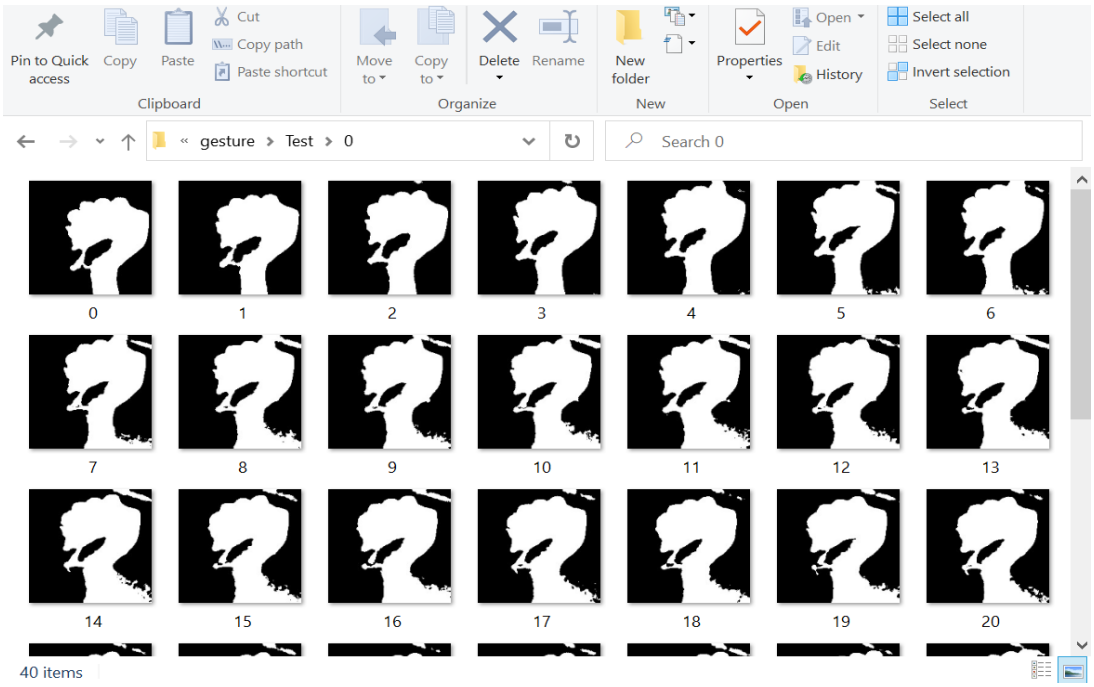
Code Snippet 1.3. Capturing RGB image input and creating thresholded images for train and test folders.



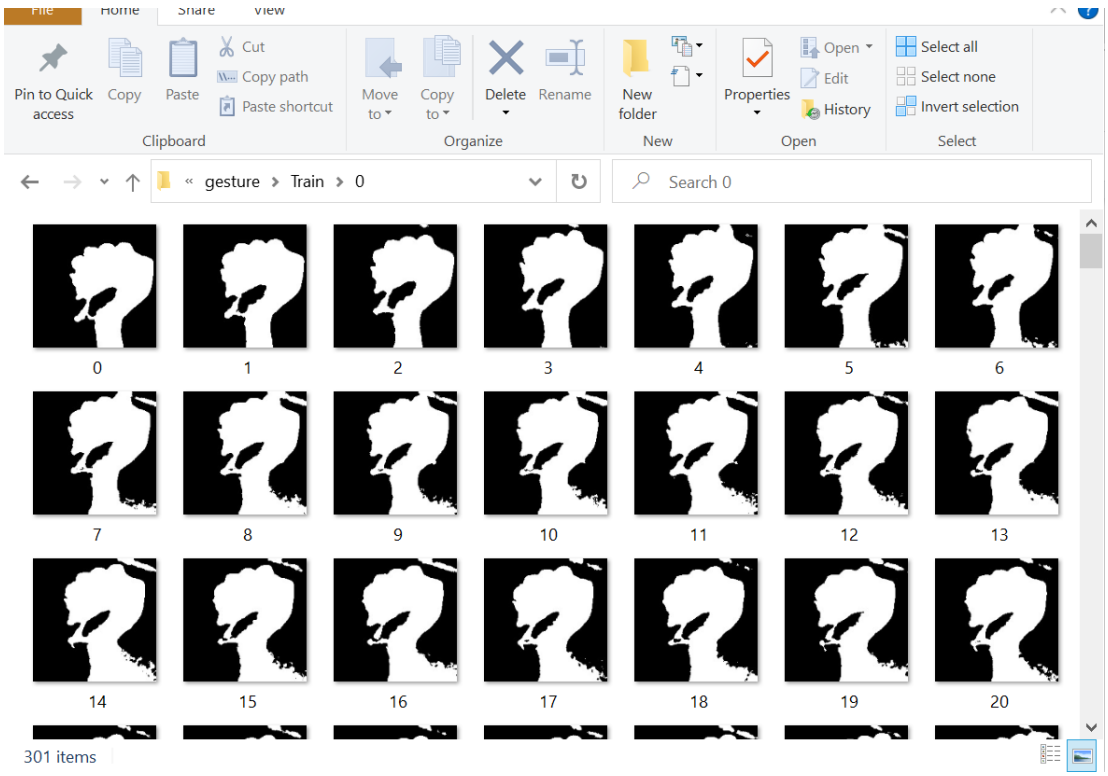
Output Screen 1.1. Background Detection



Output Screen 1.2. Thresholded hand image capturing



Output Screen 1.3. Test folder with 40 images



Output Screen 1.4. Train folder with 300 images

2. Training the CNN

C: > Users > dell > Desktop > SLR > Train_CNN.py

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dense, Flatten, BatchNormalization, Conv2D, MaxPool2D, Dropout
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from matplotlib import pyplot as plt
import itertools
import random
import warnings
import numpy as np
import cv2
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
warnings.simplefilter(action='ignore', category=FutureWarning)
train_path = r'C:\gesture\Train'
test_path = r'C:\gesture\Test'
train_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input).flow_from_directory(directory=train
test_batches = ImageDataGenerator(preprocessing_function=tf.keras.applications.vgg16.preprocess_input).flow_from_directory(directory=test_p
imgs, labels = next(train_batches)
```

Code Snippet 2.1. Importing libraries and the train and test folders

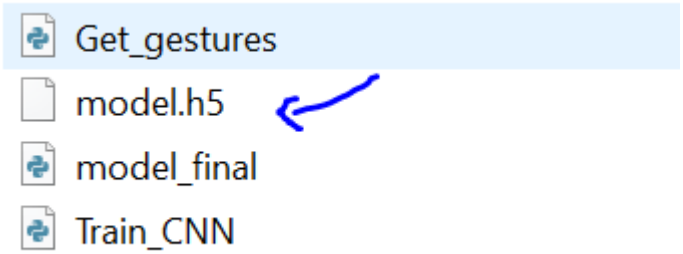
```python
model = Sequential([
    Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(64,64,3)),
    MaxPool2D(pool_size=(2, 2), strides=2), Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding = 'same'),
    MaxPool2D(pool_size=(2, 2), strides=2)
    Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding = 'valid'),
    MaxPool2D(pool_size=(2, 2), strides=2)])

model.add(Flatten())
model.add(Dense(64,activation ="relu"))
model.add(Dense(128,activation ="relu"))
model.add(Dropout(0.2))
model.add(Dense(128,activation ="relu"))
model.add(Dropout(0.3))
model.add(Dense(10,activation ="softmax"))
```

Code Snippet 2.2. Adding layers and creating the model

```python
#Compiling the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1, min_lr=0.0001)
early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, verbose=0, mode='auto')
model.compile(optimizer=SGD(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1, min_lr=0.0005)
early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, verbose=0, mode='auto')
history2 = model.fit(train_batches, epochs=10, callbacks=[reduce_lr, early_stop],  validation_data = test_batches)#, checkpoint])

imgs, labels = next(train_batches)
imgs, labels = next(test_batches)

scores = model.evaluate(imgs, labels, verbose=0)
print(f'{model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
model.save(r"model.h5")
print(history2.history)


imgs, labels = next(test_batches)
model = keras.models.load_model(r"model.h5")
scores = model.evaluate(imgs, labels, verbose=0)
print(f'{model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
model.summary()
scores
model.metrics_names
word_dict = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J'}
predictions = model.predict(imgs, verbose=0)
print("predictions on a small set of test data--")
```

```
imgs, labels = next(test_batches)
model = keras.models.load_model(r"model.h5")
scores = model.evaluate(imgs, labels, verbose=0)
print(f'{model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
model.summary()
scores
model.metrics_names
word_dict = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J'}
predictions = model.predict(imgs, verbose=0)
print("predictions on a small set of test data--")
print(" ")
for ind, i in enumerate(predictions):
    print(word_dict[np.argmax(i)], end='   ')
plotImages(imgs)
print('Actual labels')
for i in labels:
    print(word_dict[np.argmax(i)], end='   ')
print(imgs.shape)
history2.history

def plotImages(images_arr):
    fig, axes = plt.subplots(1, 10, figsize=(30,20))
    axes = axes.flatten()
    for img, ax in zip( images_arr, axes):
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        ax.imshow(img)
        ax.axis('off')
    plt.tight_layout()
    plt.show()
```

Code Snippet 2.3. Compiling the model and plotting predicted images from the test folder



Output Screen 2.1. The model is created

```
Epoch 1/10
301/301 [==============================] - 15s 47ms/step - loss: 0.5783 - accuracy: 0.9465 - val_loss: 0.0106 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 2/10
301/301 [==============================] - 14s 46ms/step - loss: 0.0030 - accuracy: 1.0000 - val_loss: 0.0045 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 3/10
301/301 [==============================] - 13s 42ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 0.0028 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 4/10
301/301 [==============================] - 13s 44ms/step - loss: 9.8893e-04 - accuracy: 1.0000 - val_loss: 0.0020 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 5/10
301/301 [==============================] - 15s 50ms/step - loss: 7.2902e-04 - accuracy: 1.0000 - val_loss: 0.0015 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 6/10
301/301 [==============================] - 16s 55ms/step - loss: 5.6941e-04 - accuracy: 1.0000 - val_loss: 0.0012 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 7/10
301/301 [==============================] - 19s 64ms/step - loss: 4.6874e-04 - accuracy: 1.0000 - val_loss: 0.0011 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 8/10
301/301 [==============================] - 18s 60ms/step - loss: 3.9499e-04 - accuracy: 1.0000 - val_loss: 8.9028e-04 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 9/10
301/301 [==============================] - 18s 60ms/step - loss: 3.3997e-04 - accuracy: 1.0000 - val_loss: 7.8644e-04 - val_accuracy: 1.0000 - lr: 0.0010
Epoch 10/10
301/301 [==============================] - 14s 46ms/step - loss: 2.9793e-04 - accuracy: 1.0000 - val_loss: 6.9242e-04 - val_accuracy: 1.0000 - lr: 0.0010
loss of 0.0008902398985810578; accuracy of 100.0%
```

Output Screen 2.2. The accuracy and loss at each epoch; Total accuracy and loss of the model

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 31, 31, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 15, 15, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 64)                294976

 dense_1 (Dense)             (None, 128)               8320

 dense_2 (Dense)             (None, 128)               16512

 dense_3 (Dense)             (None, 10)                1290

=================================================================
Total params: 414,346
Trainable params: 414,346
Non-trainable params: 0
```

Output Screen 2.3. A summary of the created model

```
Predicted Values

C   D   F   C   G   F   B   H   H   F   Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Actual Values
C   D   F   C   G   F   B   H   H   F   (10, 64, 64, 3)
```

Output Screen 2.3. Testing the prediction model o a small set of test set data. Output shows that the predicted values exactly match the actual values

3. Predicting the character

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


model = keras.models.load_model(r"model.h5")

bg_image = None
acc_weight = 0.5

rt = 200
rb = 400
rr = 200
rl = 400

word_dict = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G',7:'H',8:'I',9:'J'}


cam = cv2.VideoCapture(0,cv2.CAP_DSHOW)
num_frames =0
while True:
    ret, frame = cam.read()
    frame = cv2.flip(frame, 1)
    f_copy = frame.copy()
    roi = frame[rt:rb, rr:rl]
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (9, 9), 0)
```

Code Snippet 3.1. Loading the model and defining ROI for live gesture capture

C: > Users > dell > Desktop > SLR > model_final.py

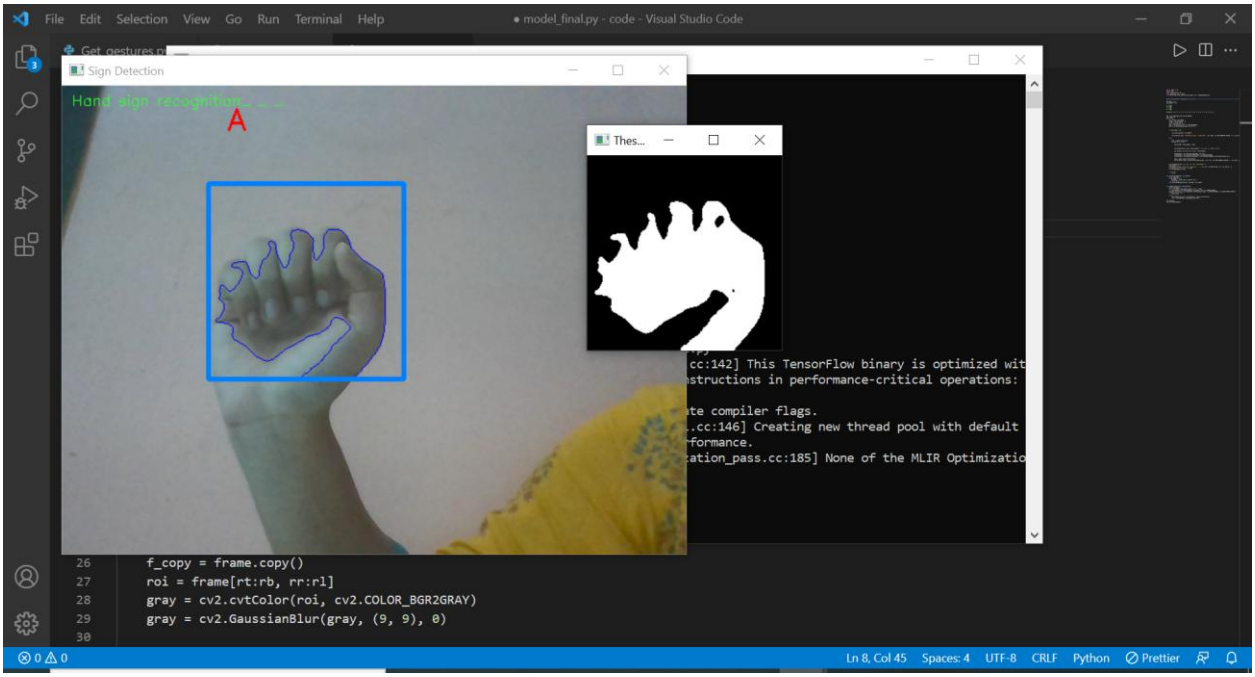
```
    if num_frames < 70:

        cal_accum_avg(gray, acc_weight)

        cv2.putText(f_copy, "FETCHING bg_image...PLEASE WAIT", (80, 400), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0,0,255), 2)

    else:

        hand = segment_hand(gray)
        if hand is not None:

            thresholded, hand_segment = hand


            cv2.drawContours(f_copy, [hand_segment + (rr, rt)], -1, (255, 0, 0),1)

            cv2.imshow("Thesholded Hand Image", thresholded)

            thresholded = cv2.resize(thresholded, (64, 64))
            thresholded = cv2.cvtColor(thresholded, cv2.COLOR_GRAY2RGB)
            thresholded = np.reshape(thresholded, (1,thresholded.shape[0],thresholded.shape[1],3))

            pred = model.predict(thresholded)
            cv2.putText(f_copy, word_dict[np.argmax(pred)], (170, 45), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)


    cv2.rectangle(f_copy, (rl, rt), (rr, rb), (255,128,0), 3)
    num_frames += 1
    cv2.putText(f_copy, "Hand sign recognition_ _ _", (10, 20), cv2.FONT_ITALIC, 0.5, (51,255,51), 1)
    cv2.imshow("Sign Detection", f_copy)
```

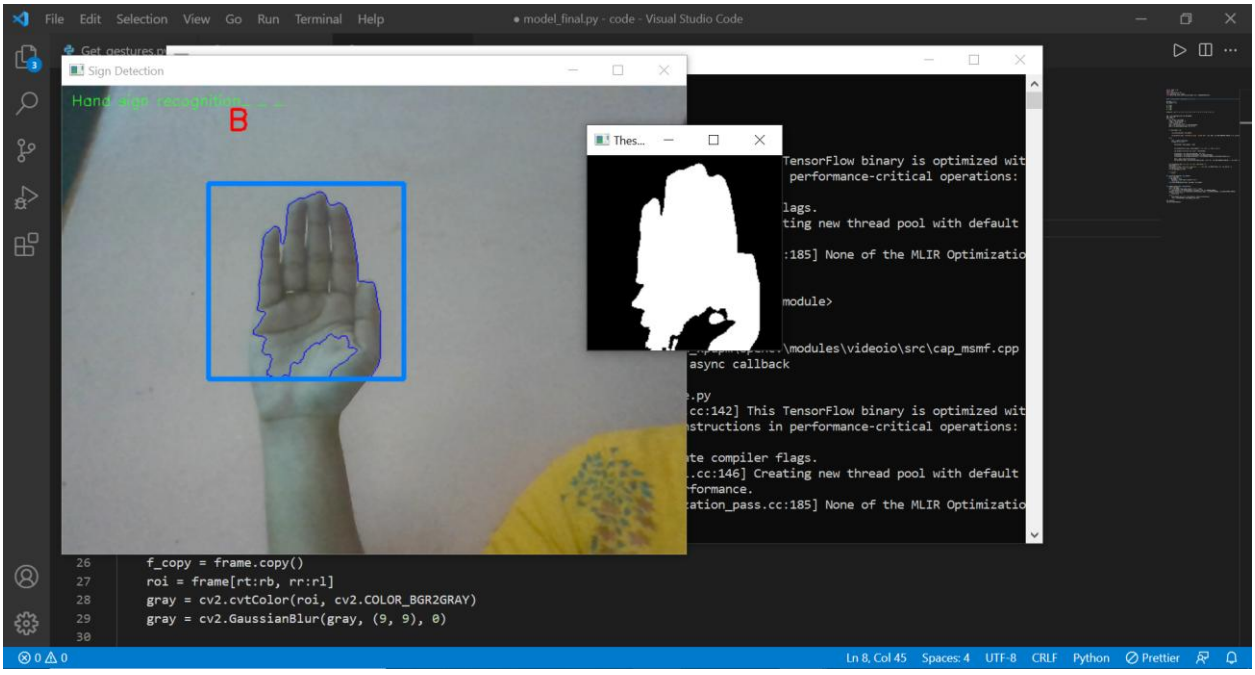
C: > Users > dell > Desktop > SLR > model_final.py

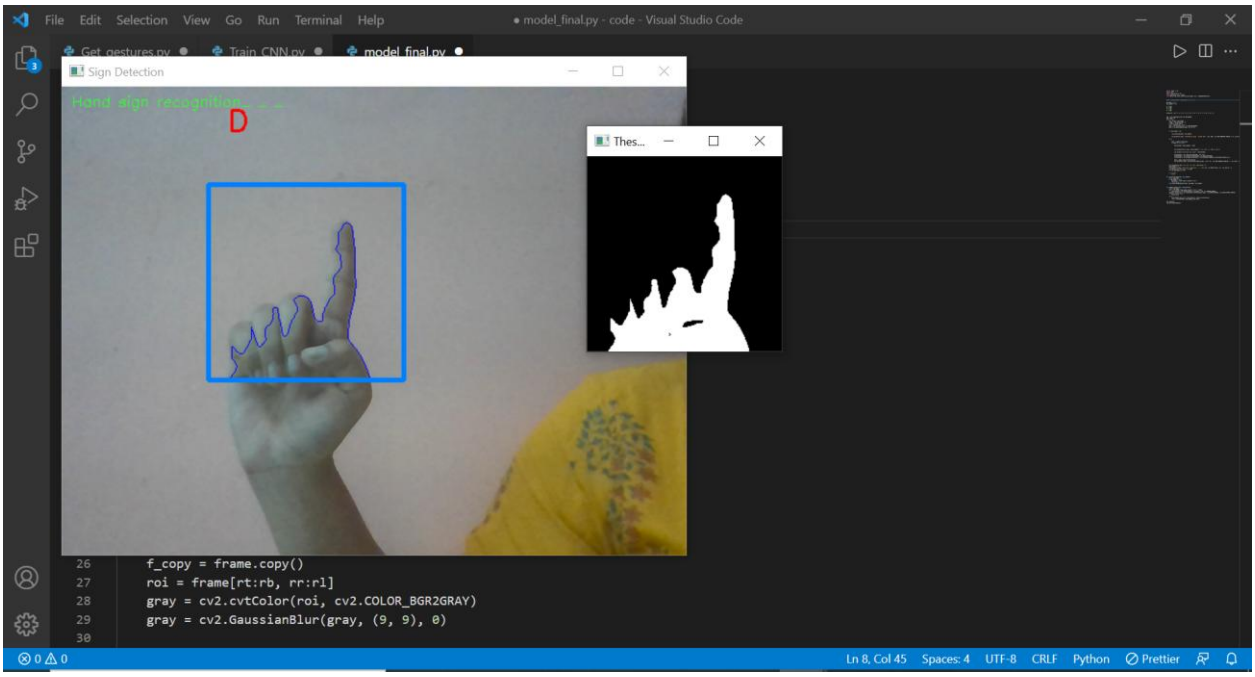
```
    cv2.rectangle(f_copy, (rl, rt), (rr, rb), (255,128,0), 3)
    num_frames += 1
    cv2.putText(f_copy, "Hand sign recognition_ _ _", (10, 20), cv2.FONT_ITALIC, 0.5, (51,255,51), 1)
    cv2.imshow("Sign Detection", f_copy)
    k = cv2.waitKey(1) & 0xFF

    if k == 27:
        break

def cal_accum_avg(frame, acc_weight):
    global bg_image
    if bg_image is None:
        bg_image = frame.copy().astype("float")
        return None
    cv2.accumulateWeighted(frame, bg_image, acc_weight)


def segment_hand(frame, threshold=25):
    global bg_image
    diff = cv2.absdiff(bg_image.astype("uint8"), frame)
    _ , thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY)
    contours, hierarchy = cv2.findContours(thresholded.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    if len(contours) == 0:
        return None
    else:
        hand_segment_max_cont = max(contours, key=cv2.contourArea)
        return (thresholded, hand_segment_max_cont)

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

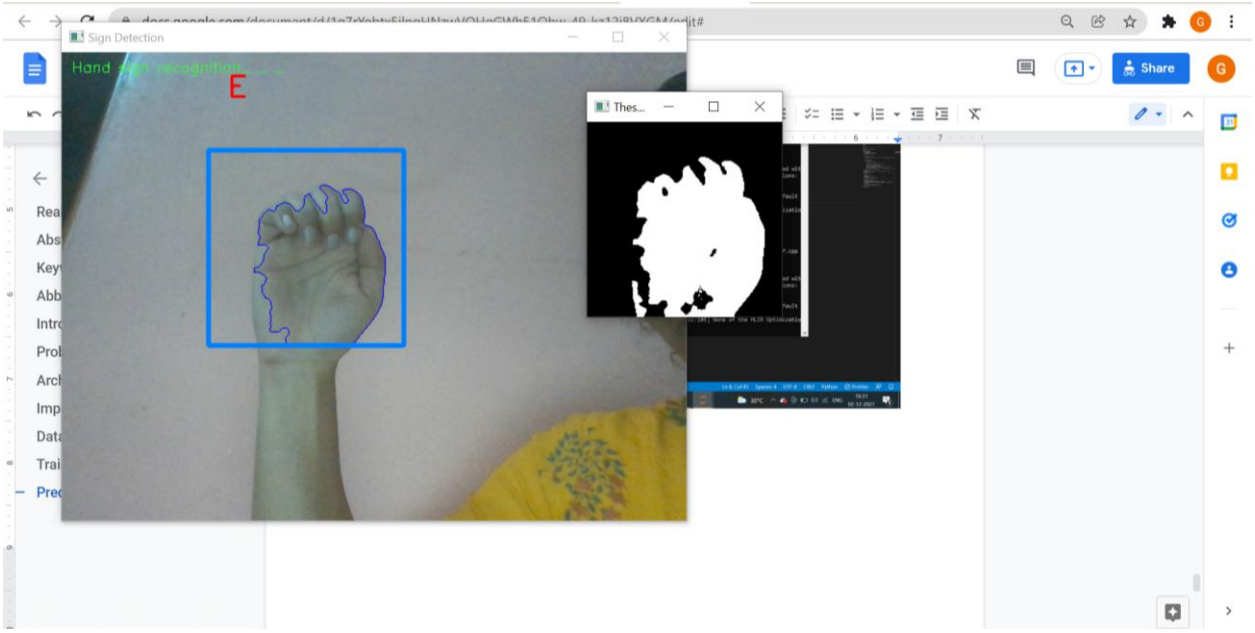
Code Snippet 3.2. Live input is thresholded and used as input for the prediction model

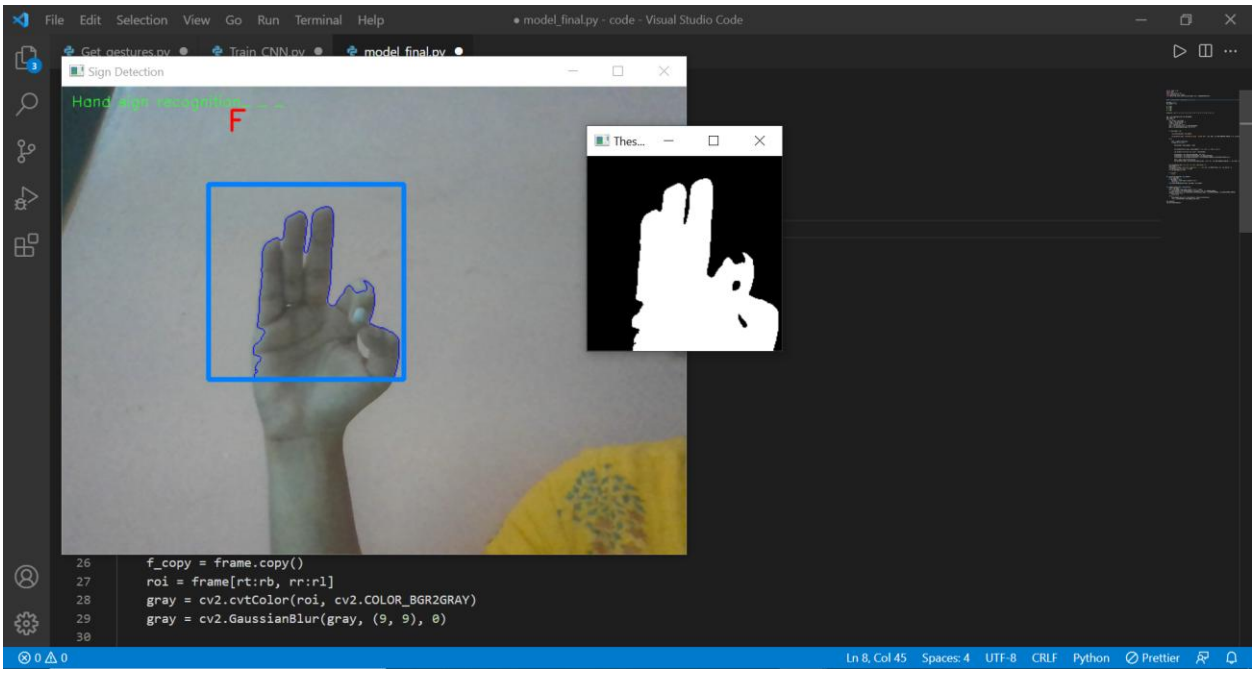
The following output screens show the model predicting the characters corresponding to the hand gesture shown in the ROI.

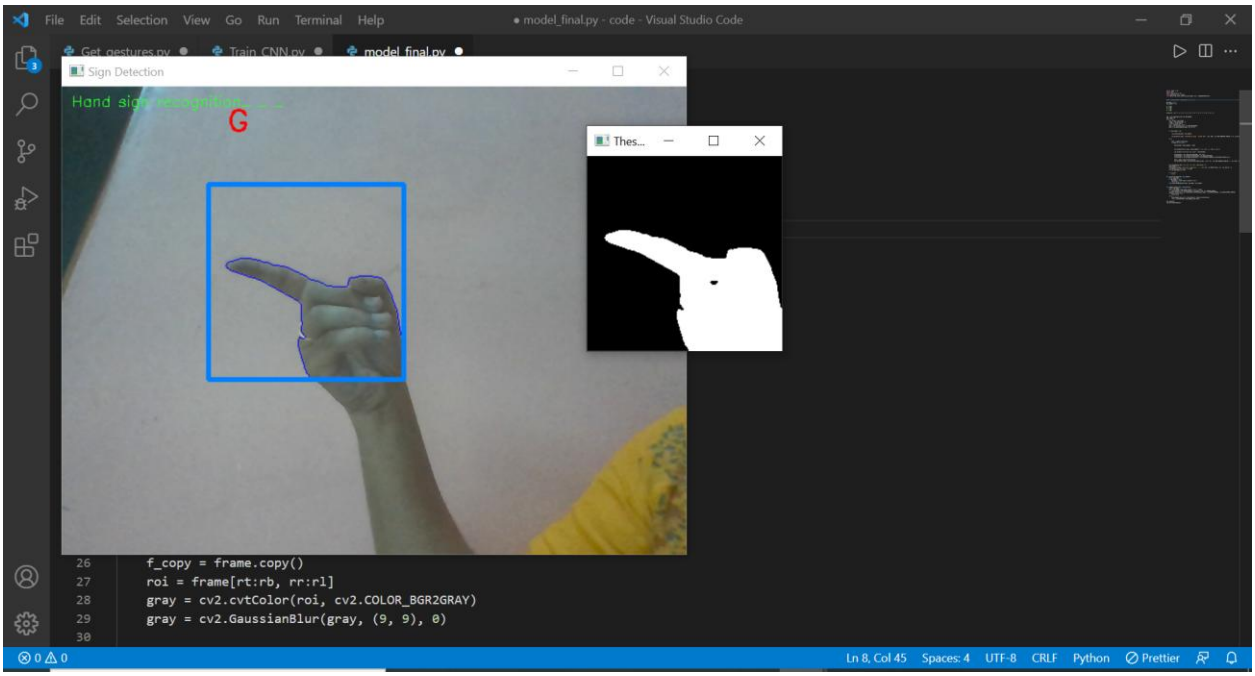


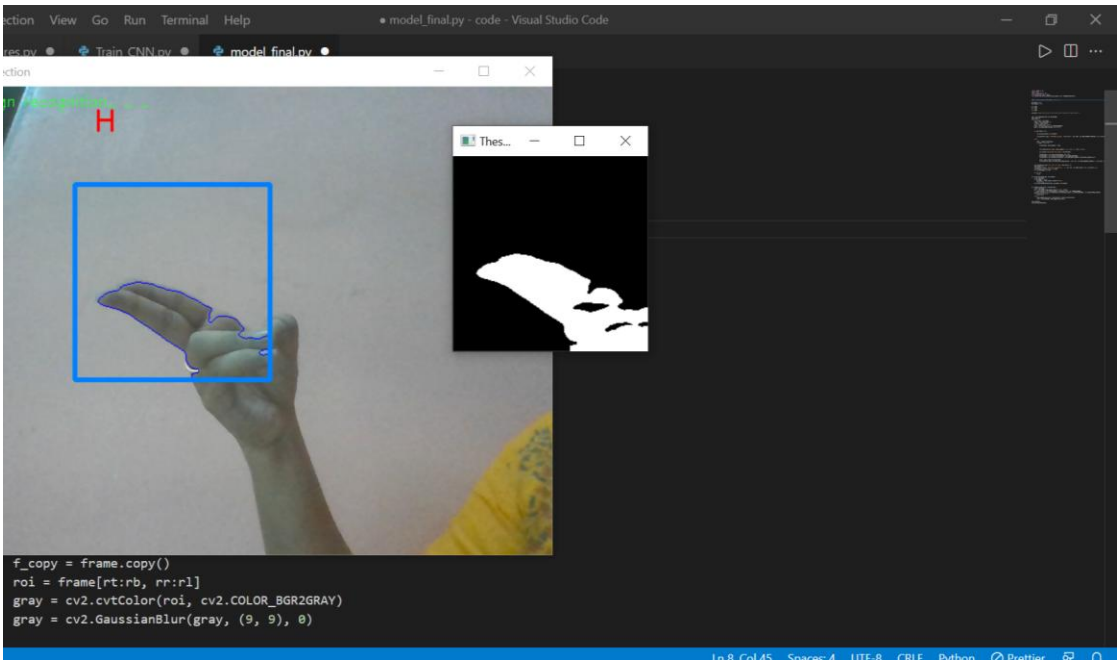


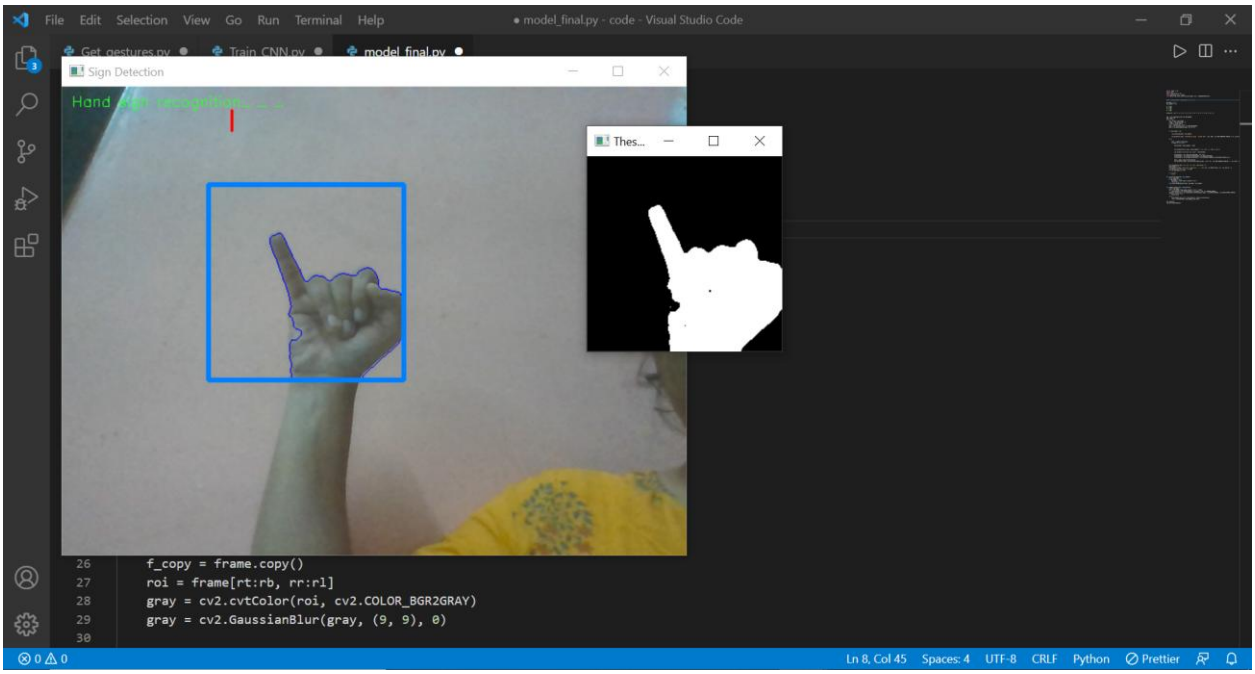

File Edit Selection View Go Run Terminal Help
• model_final.py - code - Visual Studio Code
Get_gestures.py
Train_CNN.py
model_final.py
Sign Detection
Hand sign recognition
D
Thes...
f_copy = frame.copy()
roi = frame[rt:rb, rr:rl]
gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (9, 9), 0)
Ln 8, Col 45 Spaces: 4 UTF-8 CRLF Python Prettier


Sign Detection
Hand sign recognition
E
Thes...
Share


File Edit Selection View Go Run Terminal Help
• model_final.py - code - Visual Studio Code
Get_gestures.py
Train_CNN.py
model_final.py
Sign Detection
Hand sign recognition...
F
Thes...
f_copy = frame.copy()
roi = frame[rt:rb, rr:rl]
gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (9, 9), 0)
Ln 8, Col 45 Spaces: 4 UTF-8 CRLF Python Prettier


File Edit Selection View Go Run Terminal Help
• model_final.py - code - Visual Studio Code
Get_gestures.py
Train_CNN.py
model_final.py
Sign Detection
Hand sign recognition...
G
Thes...
f_copy = frame.copy()
roi = frame[rt:rb, rr:rl]
gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (9, 9), 0)
Ln 8, Col 45 Spaces: 4 UTF-8 CRLF Python Prettier





Output Screen 3.1. The model predicts ISL gestures

IX. Conclusion and Future Work

This implementation of a Sign Language Recognition Model shows that CNNs can effectively help us recognize sign language. The model built predicts gestures with an accuracy of almost 100% and very negligible losses. Image processing methods such as contouring and segmentation were used to differentiate the foreground (hand gesture) and background, which plays an important role in determining the accuracy of the model. This model works with real time inputs and hence is a quick and efficient way of translating sign language gestures to help minimize the communication gap with the deaf community. It can further be improved to detect multiple sign languages and even build a model that will predict sentences instead of just single characters. Newer deep learning models along with transfer learning can be implemented to perform more complex functions.

X. References

1. https://towardsdatascience.com/using-ai-to-translate-sign-language-in-real-time-96fe8c8223ed
2. Singha, Joyeeta and K. Das. “Automatic Indian Sign Language Recognition for Continuous Video Sequence.” ADBU Journal of Engineering Technology (AJET) 2 (2015): n. pag.
3. Ss, Shivashankara & S, Dr.Srinath. (2018). American Sign Language Recognition System: An Optimal Approach. International Journal of Image, Graphics and Signal Processing. 10. 10.5815/ijigsp.2018.08.03.
4. Suharjito, Ricky Anderson, Fanny Wiryana, Meita Chandra Ariesta, Gede Putra Kusuma, Sign Language Recognition Application Systems for Deaf-Mute People: A Review Based on Input-Process-Output, Procedia Computer Science, Volume 116, 2017, Pages 441-448, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2017.10.028.
5. Rokade, Yogeshwar & Jadav, Prashant. (2017). Indian Sign Language Recognition System. International Journal of Engineering and Technology. 6.189-196. 10.21817/ijet/2017/v9i3/170903S030.
6. Waseem Rawat, Zenghui Wang; Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review. Neural Comput 2017; 29 (9): 2352–2449. doi: https://doi.org/10.1162/neco_a_00990
7. K. Tiku, J. Maloo, A. Ramesh and I. R., "Real-time Conversion of Sign Language to Text and Speech," 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), 2020, pp.346-351, doi: 10.1109/ICIRCA48905.2020.9182877.
8. Adewale, Victoria & Olamiti, Adejoke. (2018). Conversion of Sign Language To Text And Speech Using Machine Learning Techniques. JOURNAL OF RESEARCH AND REVIEW IN SCIENCE. 5. 58-65.10.36108/jrrslasu/8102/50(0170).