Project Report On

SIGN LANGUAGE RECOGNITION

By

Arpan Mandal
Aishwarya Dasgupta
Avi Atulya
Sampriti Chakraborty

PROJECT ABSTRACT

Sign language is the preferred method of communication among the deaf and the hearing impaired people all over the world. Recognition of sign language can have varying degree of success when used in a computer vision or any other methods. Sign language is said to have a structured set of gestures in which each gesture is having a specific meaning.

Sign Language is mainly used by deaf (hard hearing) and dumbpeople to exchange information between their own community and with other people. It is a language where people use theirhand gestures to communicate as they can't speak or hear. Sign Language Recognition (SLR) deals with recognizing the hand gestures acquisition and continues till text or speech is generated for corresponding hand gestures. Here hand gestures for sign language can be classified as static and dynamic.However, static hand gesture recognition is simpler than dynamic hand gesture recognition, but both recognition is important to the human community.We can use Deep Learning Computer Vision to recognize the hand gestures by building Deep Neural Network architectures (Convolution Neural Network Architectures) where the model will learn to recognize the hand gestures images over an epoch.Once the model Successfully recognizes the gesture the corresponding English text is generated and then text can be converted to speech.This model will be more efficient and hence communicate for the deaf (hard hearing) and dump people will beeasier. In this paper, we will discuss how Sign Language Recognition is done using Deep Learning.

The project aims at building a machine learning model that will be able to classify the various hand gestures used for fingerspelling in sign language. In this user independent model, classification machine learning algorithms are trained using a set of image data and testing is done on a completely different set of data. For the image dataset, depth images are used,

which gave better results than some of the previous literatures [4], owing to the reduced pre-processing time. Various machine learning algorithms are applied on the datasets, including Convolutional Neural Network (CNN). An attempt is made to increase the accuracy of the CNN model by pre-training it on the Imagenet dataset. However, a small dataset was used for pre-training, which gave an accuracy of 15% during training.

CONTENTS

CHAPTER 8: REFERENCES

INTRODUCTION

The project aims at building a machine learning model that will be able to classify the various hand gestures used for fingerspelling in sign language. In this user independent model, classification machine learning algorithms are trained using a set of image data and testing is done on a completely different set of data. For the image dataset, depth images are used, which gave better results than some of the previous literatures, owing to the reduced pre-processing time. Various machine learning algorithms are applied on the datasets, including Convolutional Neural Network (CNN). An attempt is made to increase the accuracy of the CNN model by pre-training it on the ImageNet dataset. However, a small dataset was used for pre-training.

Deaf (hard hearing) and dumb people use Sign Language (SL) as their primary means toexpress their ideas and thoughtswith their own community and with other people with hand and body gestures.It has its own vocabulary, meaning, and syntax which is different from the spoken language or written language. Spoken language is a language produced by articulate sounds mapped against specific words and grammatical combinations to convey meaningful messages. Sign language uses visual hand and body gestures to convey meaningful messages.There are somewhere between 138 and 300 different types of Sign Language used around globally today. In India, there are only about 250 certified sign language interpreters for a deaf population of around 7 million. This would be a problem to teach sign language to the deaf and dumb people as there isa limitednumber of sign language interpreters exits today.Sign Language Recognition is an attempt to recognize these hand gestures and convert them to the corresponding text or speech.Today Computer Vision and Deep Learning have gained a lot of popularity and many State of the Art (SOTA) models can be built. Using Deep Learning algorithms and Image Processing we can able to classify these hand gestures and able to produce corresponding text.An example of “A” alphabet in sign language notion to English “A” text or speech.

1.1. Objective:

- Sign Language consists of fingerspelling, which spells out words character by character, and word level association which involves hand gestures that convey the word meaning.
- Fingerspelling is a vital tool in sign language, as it enables the communication of names, addresses and other words that do not carry a meaning in word level association.
- In spite of this, fingerspelling is not widely used as it is challenging to understand and difficult to use.
- Moreover, there is no universal sign language and very few people know it, which makes it an inadequate alternative for communication.
- A system for sign language recognition that classifies finger spelling can solve this problem. Various machine learning algorithms are used and their accuracies are recorded and compared in this report.

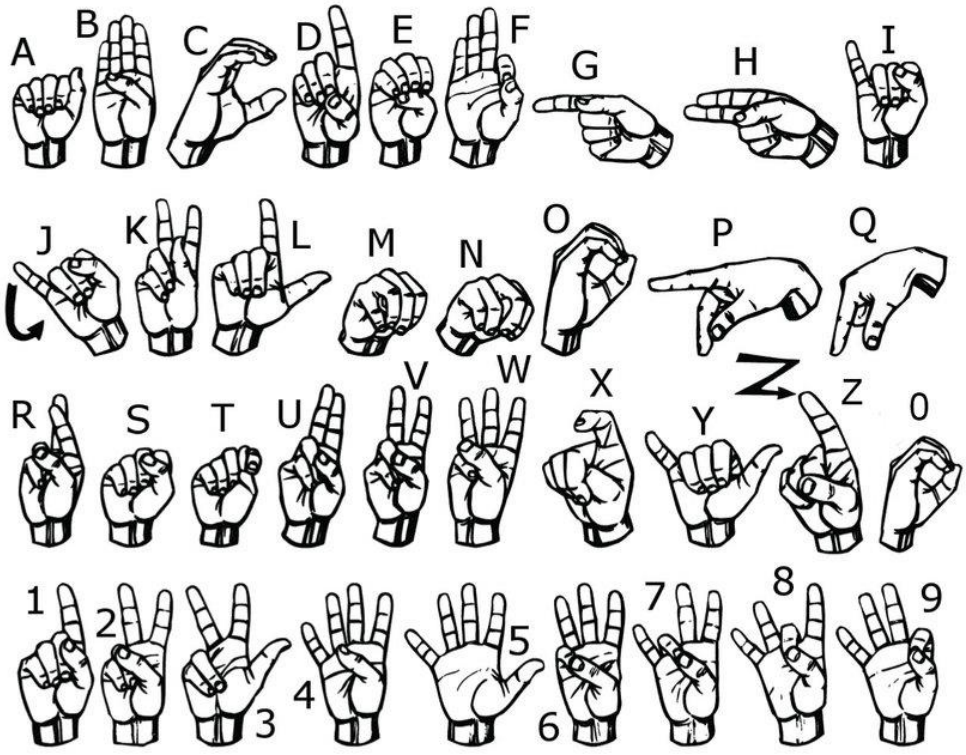

Fig 1.1.1: Sign Language Hand Gestures

1.2. Scope of the system:

Sign Language chiefly uses manual communication to convey meaning. This involves simultaneously combining hand shapes, orientations and movement of the hands, arms or body to express the speaker's thoughts.

However, unfortunately, for the speaking and hearing impaired minority, there is a communication gap. Visual aids, or an interpreter, are used for communicating with them.

1.3. Feasibility Study:

1.3.1. Technical Feasibility:

We have used certain technologies for this project, named as follows:

- Python
- Numpy
- Tensorflow
- OpenCv
- Keras
- Teachable Machine

Each of these technologies are freely available and the technical skills required are manageable.

So from all this it is clear that this project is technically feasible.

1.3.2. Operational Feasibility:

This project is completely made for the people who use sign language as a mode of communication as they are not able to speak. So, for us who don’t understand these symbols, it is henceforth a difficult task to communicate with them.

This project proposes a model so that the sign and symbols that are used to communicate can be converted easily into text and we can derive the meaning from it.

So this project is operationally feasible too.

Our model detects each of the alphabet in real time without any delay, which makes it easier for the user to understand the meaning of the signs.

1.3.3. Economic Feasibility:

This project will be freely available for potential customers that means no cost will be charged from them. As there is nothing paid resource and technologies used in the product so for now it is free but in future we are going to use a larger database which will required paid version of database. But for that too no any cost will be charged from potential customers.

So from this it is clear that the project is economically feasible as well.

SRS (Software Requirements Specification)

2.1. Introduction:

The project aims at building a machine learning model that will be able to classify the various hand gestures used for fingerspelling in sign language. In this user independent model, classification machine learning algorithms are trained using a set of image data and testing is done on a completely different set of data. For the image dataset, depth images are used, which gave better results than some of the previous literatures, owing to the reduced pre-processing time. Various machine learning algorithms are applied on the datasets, including Convolutional Neural Network (CNN). An attempt is made to increase the accuracy of the CNN model by pre-training it on the ImageNet dataset. However, a small dataset was used for pre-training.

2.1.1. Purpose:

Communication is very crucial to human beings, as it enables us to express ourselves. We communicate through speech, gestures, body language, reading, writing or through visual aids, speech being one of the most commonly used among them. However, unfortunately, for the speaking and hearing impaired minority, there is a communication gap. Visual aids, or an interpreter, are used for communicating with them. However, these methods are rather cumbersome and expensive, and can't be used in an emergency. Sign Language chiefly uses manual communication to convey meaning. This involves simultaneously combining hand shapes, orientations and movement of the hands, arms or body to express the speaker's thoughts.

2.1.2. Intended Audience:

People who have hearing and speech impairment.

2.1.3. Intended Use:

It is used to communicate to hearing and speech impaired people with ease.

2.1.4. Scope:

As we normally speak to each other to communicate and convey our messages but for the speech and hearing-impaired people it is difficult to communicate. Using this they can easily communicate with others who doesn't know sign language. As learning to use sign languages is a time consuming process this will help speed up the process.

2.2. Overall Description:

2.2.1. User's Need:

1. A working webcam to capture the images to run in the system.

2.2.2. Assumption and Dependencies:

Assumption:

1. Adequate storage space and webcam access should be there.
2. Database will not make any problem.

Dependencies:

1. Good quality webcam so that the images should be clear.

2.3. System Features and Requirements:

2.3.1. Functional Requirements:

1. OpenCV is used to capture the images

2. Python is required to run the program.

3. Keras is used to train the model and predict the outcome.

2.3.2. System Features :

1. Can run on any OS, i.e., Windows, Redhat etc. There are no any restrictions on using any window.

2. Only require a Desktop or a laptop with a active working webcam.

SOFTWARE DEVELOPMENT PROCESS MODEL ADOPTED

Incremental Model

Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release.
The process continues until the complete system achieved.

The various phases of Incremental model are as follows:

1. Requirement analysis: In the first phase of the incremental model, the product analysis expertise identifies the requirements. And the system functional requirements are understood by the requirement analysis team. To develop the software under the incremental model, this phase performs a crucial role.
2. Design & Development: In this phase of the Incremental model of SDLC, the design of the system functionality and the development method are finished with success. When software develops new practicality, the incremental model uses style and development phase.
3. Testing: In the incremental model, the testing phase checks the performance of each existing function as well as additional functionality. In the testing phase, the various methods are used to test the behavior of each task.
4. Implementation: Implementation phase enables the coding phase of the development system. It involves the final coding that design in the designing and development phase and tests the functionality in the testing phase. After completion of this phase, the number of the product working is enhanced and upgraded up to the final system product

When we use the Incremental Model?

1. When the requirements are superior.
2. A project has a lengthy development schedule.
3. When Software team are not very well skilled or trained.
4. When the customer demands a quick release of the product.
5. You can develop prioritized requirements first.
6. Advantage of Incremental Model
7. Errors are easy to be recognized.
8. Easier to test and debug
9. More flexible.
10. Simple to manage risk because it handled during its iteration.
11. The Client gets important functionality early.
12. Disadvantage of Incremental Model
13. Need for good planning
14. Total Cost is high.
15. Well defined module interfaces are needed.

OVERVIEW

4.1 SYSTEM OVERVIEW

Sign Language consists of fingerspelling, which spells out words character by character, and word level association which involves hand gestures that convey the word meaning. Fingerspelling is a vital tool in sign language, as it enables the communication of names, addresses and other words that do not carry a meaning in word level association. In spite of this, fingerspelling is not widely used as it is challenging to understand and difficult to use. Moreover, there is no universal sign language and very few people know it, which makes it an inadequate alternative for communication.

4.1.1. LIMITATION OF EXISTING SYSTEM

- Sign Language Recognition System has been developed from classifying only static signs and alphabets, to the system that can successfully recognize dynamic movements that comes in continuous sequences of images.

4.2. PROPOSED SYSTEM

- PyCharm

4.2.1. OBJECTIVES OF THE PROPOSED SYSTEM

- PyCharm – To connect the images of test and train dataset and predicting the output efficiently.

4.2.2. USERS OF THE PROPOSED SYSTEM

- It will be used by the single user.

TECHNOLOGIES

In this chapter we will discuss the tools we have to use to implement the project, development environment, software interface and hardware which will be required for successful completion of the project. Now we will discuss them one by one

5.1. Tools used in development:

1. Python
2. OpenCV
3. Keras
4. Tensorflow
5. Numpy
6. Teachable Machine

5.2. Development Environment:

1. PyCharm

5.3. Hardware Used:

1. Computer with 64-bit processor and four-core, 2.5 GHz minimum per core
2. RAM – 4 GB or Higher
3. Storage – 256 GB or Higher HDD (or SSD)
4. Input Device – Keyboard, Mouse, Webcam
5. Output Device – Monitor

5.4. Working of the System

5.4.1 Sign Language Gestures Images Dataset

Sign Language Gesture Images Dataset The dataset consists of 26 different hand sign gestures which include A-Z alphabet gestures. Each gesture has 1500 images which are 50x50 pixels, so altogether there are 37 gestures which means there 55,500 images

for all gestures. Convolutional Neural Network (CNN) is well suited for this dataset for model training purposes and gesture prediction.

5.4.2 Data Pre Processing

An image is nothing more than a 2-dimensional array of numbers or pixels which are ranging from 0 to 255.Typically, 0 means black, and 255 means white. Image is defined by mathematical function f(x,y) where ‘x’ represents horizontal and ‘y’ represents vertical in a coordinate plane. The value of f(x, y) at any point is giving the pixel value at that point of an image. Image Pre-processing is the use of algorithms to perform operations on images. It is important to Pre-process the images before sending the images for model training. For example, all the images should have the same size of 200x200 pixels. If not,the model cannot be trained.

Fig 5.4.1: Image without Pre Processing

Fig 5.4.2: Processed Image

The steps we have taken for image Pre-processing are:

- Read Images.
- Resize or reshape all the images to the same.
- Remove noise.
- All the image pixels arrays are converted to 0 to 255 by dividing the image array by 255.

5.4.3 Training of Model

To train the model we've used teachable machine. Teachable Machine is a web-based tool that makes creating machine learning models fast, easy, and accessible to everyone. Teachable Machine is flexible – use files or capture examples live. It's respectful of the way you work. You can even choose to use it entirely on-device, without any webcam or microphone data leaving your computer.

We have made 26 different classes for each alphabets and exported the caotured images of the signs to each respective alphabet classes. The training model uses 50 epochs and learning rate of 0.001. The keras trained model can be obtained using that.

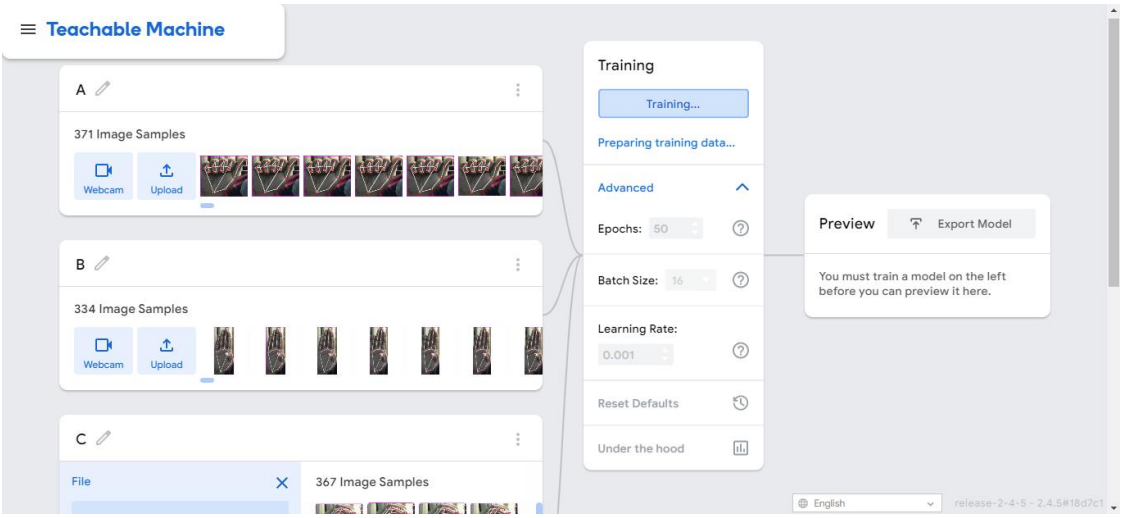

Fig 5.4.3: Training the model

As the screenshot describes, different classes and different set of images are exported and then trained.

The steps we have taken for training the model:

- Creating classes
- Exporting Images
- Setting up epoch size and train.

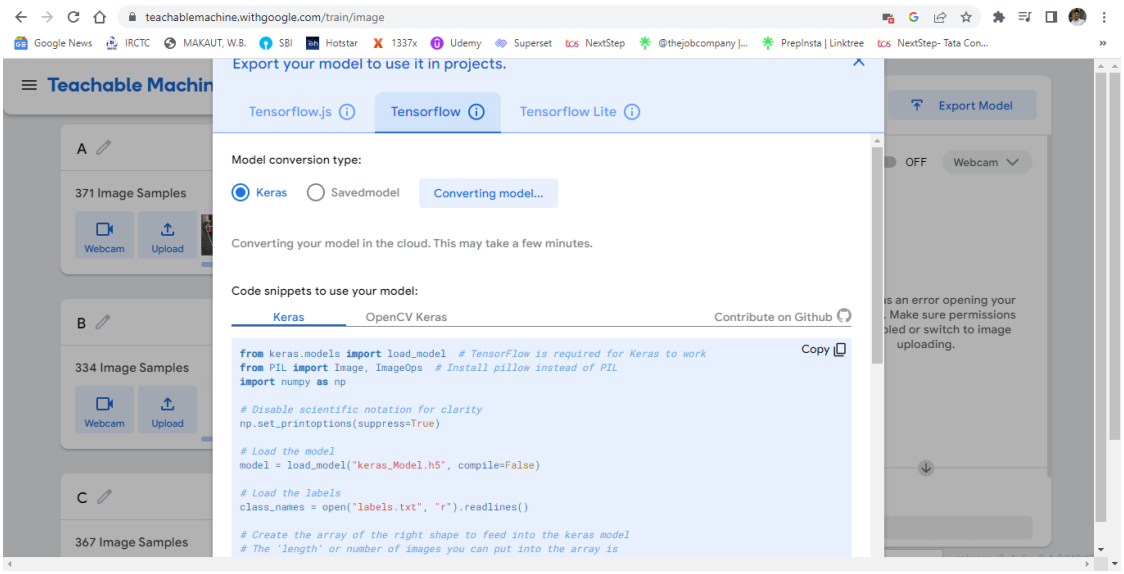

Fig 5.4.3: Downloading the trained keras model

5.4.4 Code for Data Collection

```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
import math
import time

cap = cv2.VideoCapture(0,cv2.CAP_DSHOW)
detector = HandDetector(maxHands=1)

offset = 20
imgSize = 300

folder = "Data/Z"
counter = 0

while True:
    success, img = cap.read()
    hands, img = detector.findHands(img)
    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']

        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
        imgCrop = img[y - offset:y + h + offset, x - offset:x + w +
offset]

        imgCropShape = imgCrop.shape

        aspectRatio = h / w

        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            imgResizeShape = imgResize.shape
```

```
            wGap = math.ceil((imgSize - wCal) / 2)
            imgWhite[:, wGap:wCal + wGap] = imgResize

        else:
            k = imgSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop,(imgSize, hCal))
            imgResizeShape = imgResize.shape
            hGap = math.ceil((imgSize - hCal) / 2)
            imgWhite[hGap:hCal + hGap, :] = imgResize

        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)

    cv2.imshow("Image", img)
    key = cv2.waitKey(1)
    if key == ord("s"):
        counter += 1
        cv2.imwrite(f'{folder}/Image_{time.time()}.jpg',imgWhite)
        print(counter)
```

5.4.4 Code for Testing the data

```
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
import math

cap = cv2.VideoCapture(0,cv2.CAP_DSHOW)
detector = HandDetector(maxHands=1)
classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")

offset = 20
imgSize = 300

folder = "Data/C"
counter = 0

labels = ["A", "B", "C","D", "E", "F","G", "H", "I","J", "K",
"L","M", "N", "O","P", "Q", "R","S", "T", "U","V", "W", "X","Y", "Z"]

while True:
    success, img = cap.read()
    imgOutput = img.copy()
    hands, img = detector.findHands(img)
    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']

        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
        imgCrop = img[y - offset:y + h + offset, x - offset:x + w +
offset]

        imgCropShape = imgCrop.shape

        aspectRatio = h / w

        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
```

```
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            imgResizeShape = imgResize.shape
            wGap = math.ceil((imgSize - wCal) / 2)
            imgWhite[:, wGap:wCal + wGap] = imgResize
            prediction, index = classifier.getPrediction(imgWhite,
draw=False)
            print(prediction, index)

        else:
            k = imgSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))
            imgResizeShape = imgResize.shape
            hGap = math.ceil((imgSize - hCal) / 2)
            imgWhite[hGap:hCal + hGap, :] = imgResize
            prediction, index = classifier.getPrediction(imgWhite,
draw=False)

        cv2.rectangle(imgOutput, (x - offset, y - offset-50),
                      (x - offset+90, y - offset-50+50), (255, 0,
255), cv2.FILLED)
        cv2.putText(imgOutput, labels[index], (x, y-26),
cv2.FONT_HERSHEY_COMPLEX, 1.7, (255, 255, 255), 2)
        cv2.rectangle(imgOutput, (x-offset, y-offset),
                      (x + w+offset, y + h+offset), (255, 0, 255), 4)
        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)

    cv2.imshow("Image", imgOutput)
    cv2.waitKey(1)
```

DESIGN

In this chapter Data Flow Diagram will be described

6.1. Data Flow Diagram:

We will describe Level 0, Level 1 and Level 2 DFD here

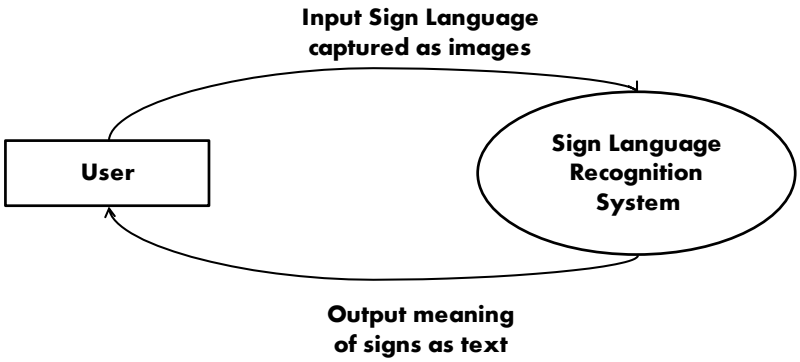

Fig. 6.1.1: Level 0 DFD

As we know in level 0 DFD only one process should be there. The user will interact with the model. The input will be the signs and symbols, and the output will be predicted by the system which is text.

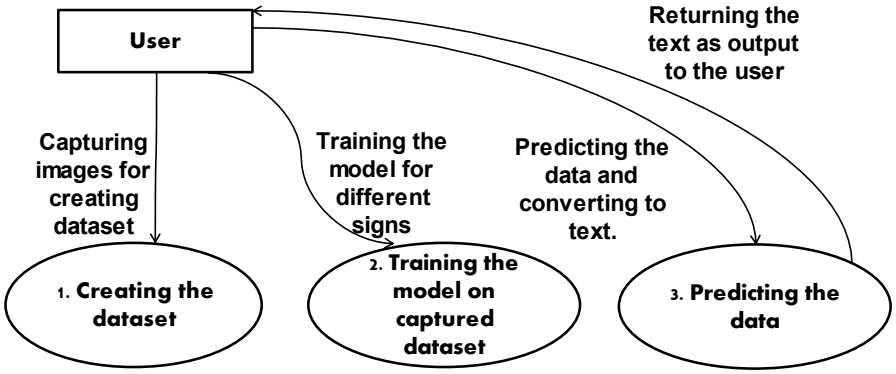

Fig. 6.1.2: Level 1 DFD

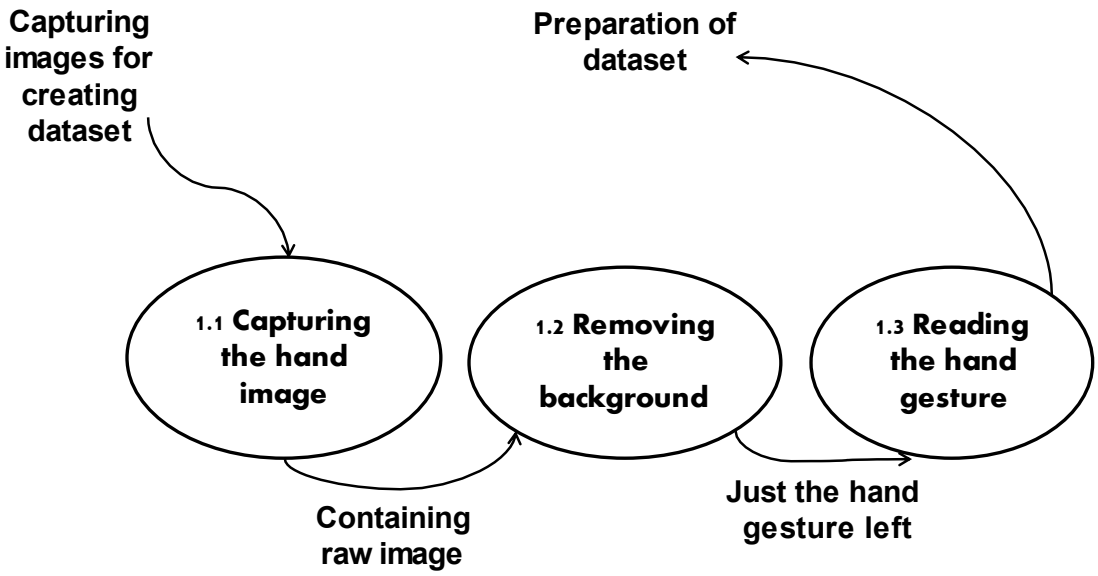

Fig. 6.1.3: Level 2 DFD

SNAPSHOTS

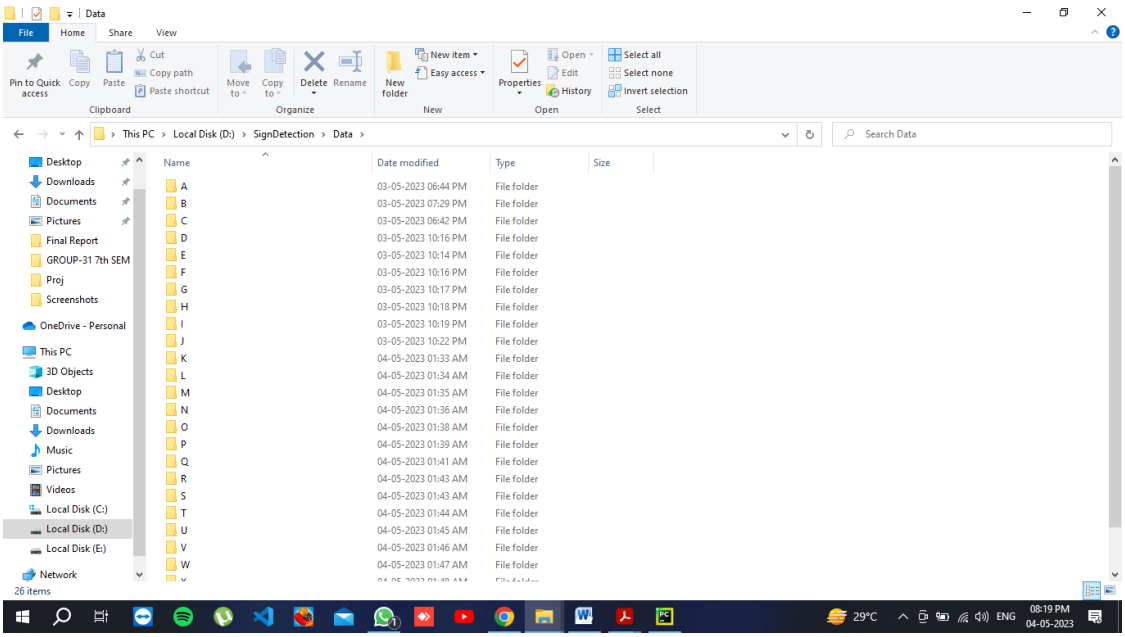

Fig 7.1: Directory for storing the train dataset

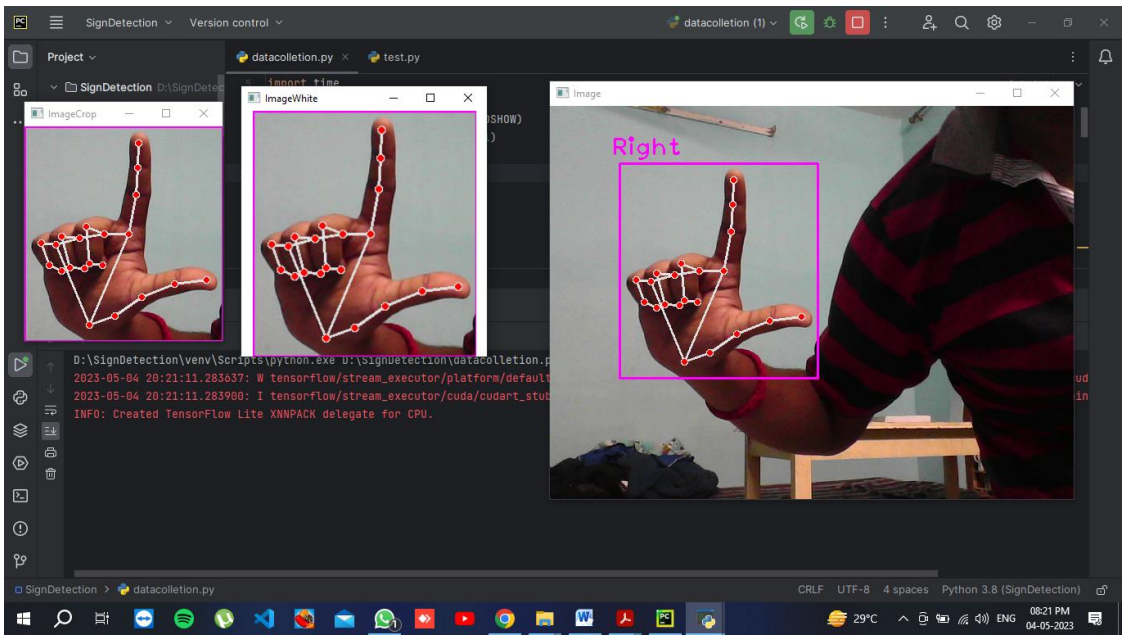

Fig 7.2: ROI for capturing images (blue square indicates the ROI)

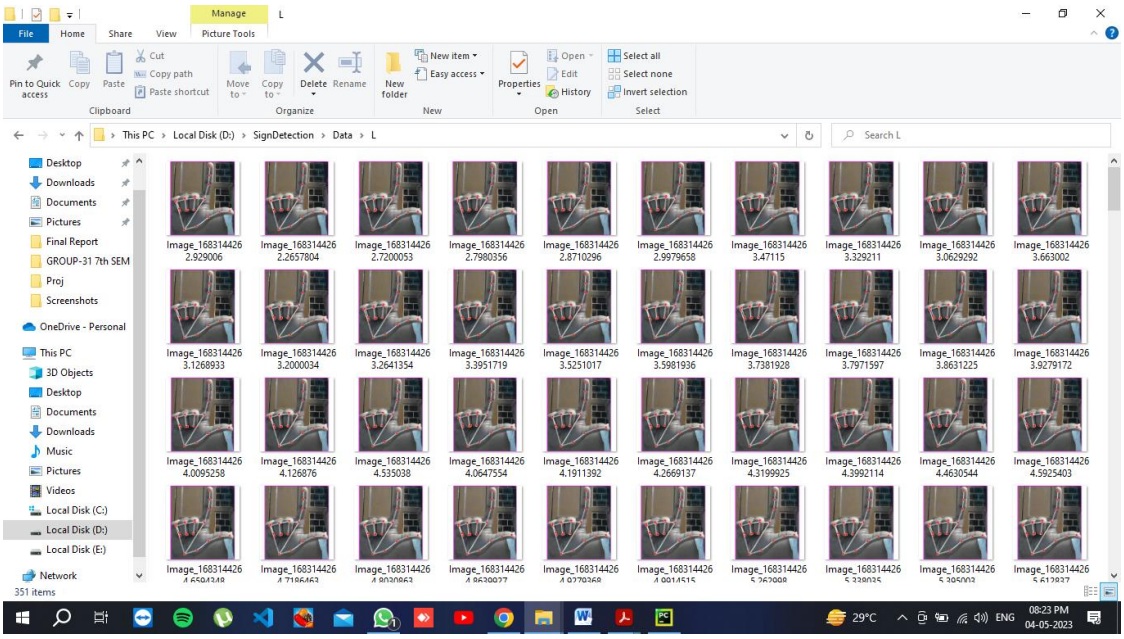

Fig 7.3: Captured images

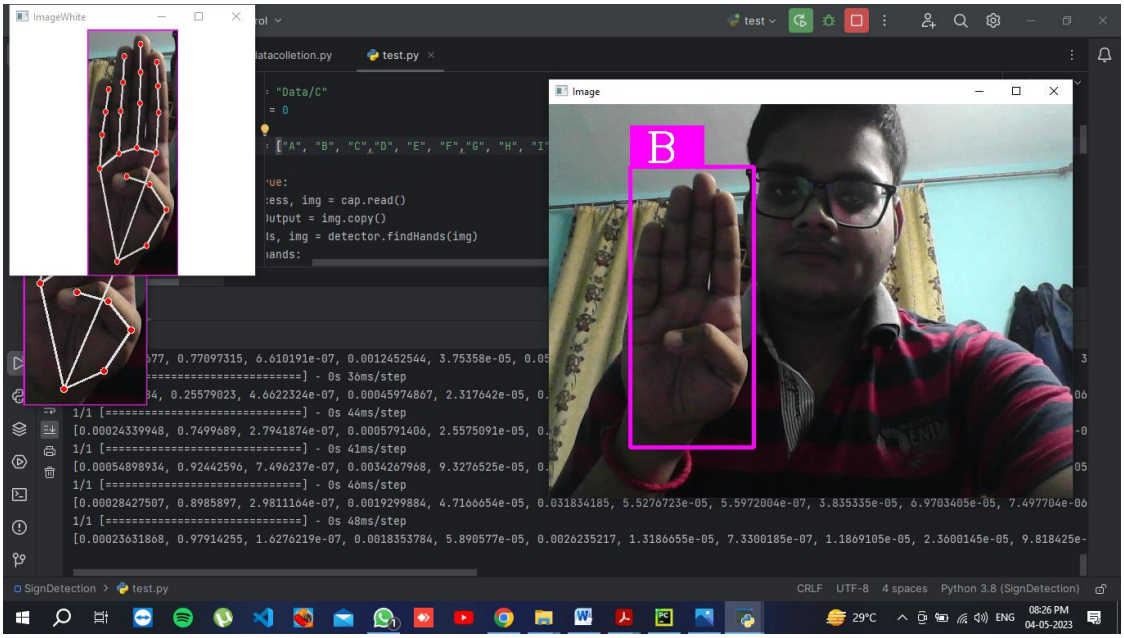

Fig 7.4: Predicting the sign as ‘B’

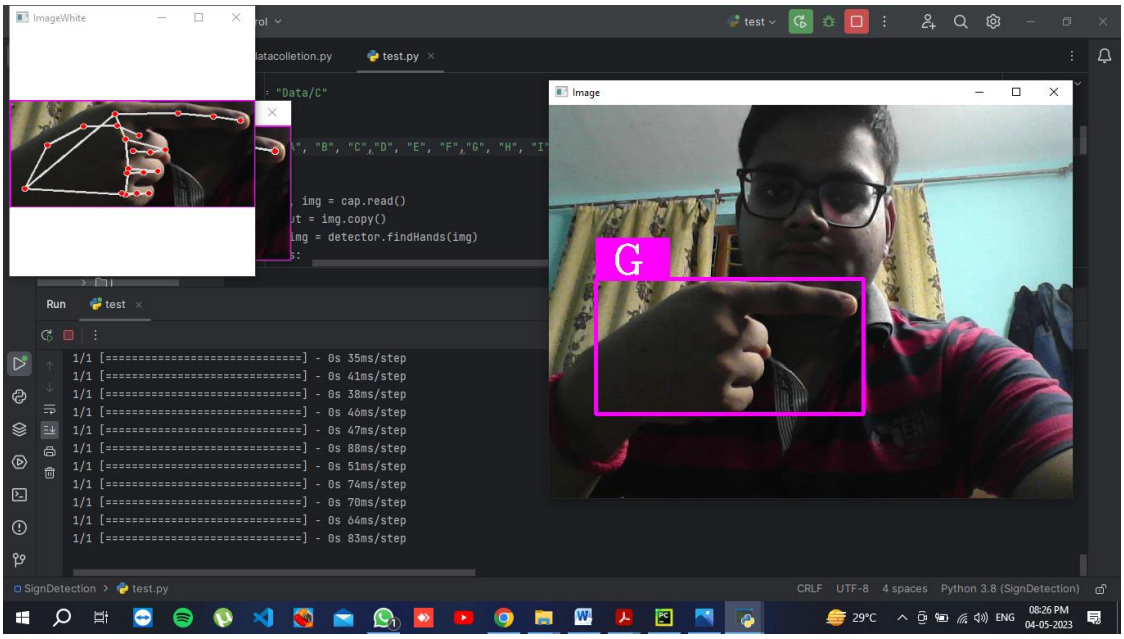

Fig 7.5: Predicting the sign as ‘G’

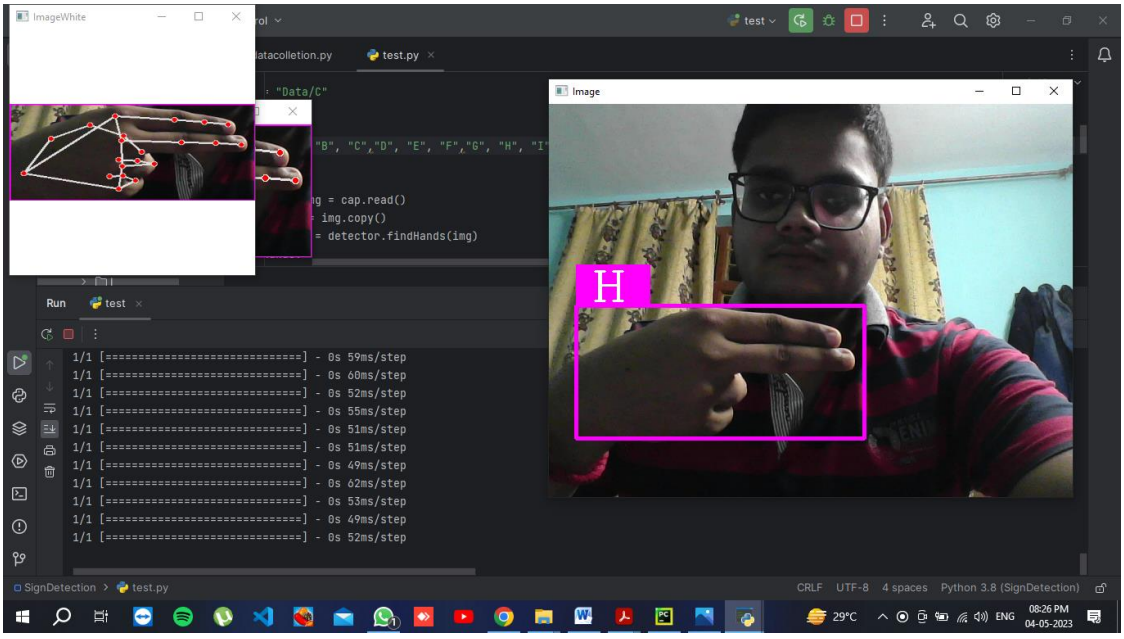

Fig 7.6: Predicting the sign as ‘H’

REFERENCES

1. ‘Programming in Java’ by Sachin Malhotra & Saurabh Choudhary – 2013 edition pp. 256 – 285

2. ‘Android Programming’ by Bill Phillips & Brian Hardy – 2013 edition pp. 96 – 111, pp. 136 – 147, 214 – 223, pp. 397 – 400, pp. 401 – 420

3. ‘Head First Android Development’ by David Griffths & Dawn Griffiths – 2015 edition pp. 438 – 480, pp. 517 – 584

4. ‘Head First PHP & MySQL’ by Lynn Beighley & Michael Morrison – 2009 edition pp. 59 – 102, pp. 120 – 158, pp. 501 – 560