A Midterm Progress Report

On

SIGN LANGUAGE RECOGNITION

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

Computer Science and Engineering

SUBMITTED BY

NIMISH BUDHIRAJA	SAHIL SHARMA	SANYAM GARG
2004631	2004657	2004659

UNDER THE GUIDANCE OF

DR. KAPIL SHARMA

(JAN – MAY 2024)

Department of Computer Science and Engineering

GURU NANAK DEV ENGINEERING COLLEGE, LUDHIANA

INDEX

1. Introduction

1.1 Introduction of the project

Sign Language Recognition focuses on developing software or hardware solutions that can recognize and interpret sign language gestures and movements made by individuals with hearing or speech impairments. Sign language is a visual language that uses a combination of hand gestures, facial expressions, and body language to convey information and meaning. It is used by millions of people worldwide as their primary means of communication. However, for those who do not understand sign language, communication with individuals who use it can be challenging.

The goal of sign language recognition is to bridge this communication gap by developing technology that can accurately interpret sign language gestures and translate them into spoken or written language. This can help individuals who use sign language to communicate more effectively with those who do not.

There are several approaches to sign language recognition, including computer vision and machine learning techniques. These methods involve training models to recognize and interpret different sign language gestures and movements based on visual data. With the help of these techniques, it is possible to create systems that can recognize a wide range of sign language gestures and translate them into spoken or written language in real-time.

1.2 Objectives of the project

- To recognize hand gestures which include 26 English alphabets (A-Z) and 10 digits (0-9) using Convolutional Neural Network.
- To convert sign language into words by an algorithm or a model.
- To show on optical viewfinder of camera module what a particular position of hand means with respect to sign language.

2. System Requirements

2.1 Software Requirements

- **Programming Languages:** Knowledge of programming languages such as Python will be required to develop the software for Sign Language Recognition.
- **Development Environments:** Visual Studio Code, or Eclipse may be required for software development.
- **Libraries and Frameworks:** Machine learning and computer vision libraries and frameworks such as TensorFlow, Keras and OpenCV will be needed to develop models and algorithms for Sign Language Recognition.
- **Datasets:** Access to Sign Language Recognition datasets such as ASL Alphabet Dataset is required to train the models and algorithms.

2.2 Hardware Requirements

- **Web Camera:** A high-quality web camera is required to capture the gestures and movements made by the individuals using sign language.
- **Processing Power:** Sign Language Recognition may require high processing power, especially for real-time recognition. Thus, a powerful processor, such as an Intel Core i5 or higher, or a dedicated graphics card may be needed.
- **Memory:** Sufficient RAM, such as 8 GB or more, will be required to handle the large datasets and models used for Sign Language Recognition.

3. Software Requirement Analysis

3.1 Define the problem

Sign Language Recognition is a problem of interpreting sign language gestures and movements made by individuals with speech impairments and translating them into written language. The goal of this problem is to create technology that can help bridge the communication gap between sign language users and those who do not understand it.

The Sign Language Recognition problem can be approached using different techniques, including computer vision and machine learning. The computer vision approach involves analyzing and processing the visual data of the sign language gestures captured by a camera. On the other hand, the machine learning approach involves training models to recognize different sign language gestures and movements based on the data input.

3.2 Define the modules and their functionalities

- **NumPy:** NumPy is a Python library used for numerical computing. It provides a wide range of mathematical functions and tools for working with arrays and matrices.
- **OpenCV:** OpenCV (Open Source Computer Vision Library) is a popular computer vision library that provides a range of algorithms and techniques for image processing, including image filtering, edge detection, feature extraction, and object detection. In sign language recognition projects, OpenCV is often used for hand detection and tracking, as well as feature extraction from hand images.
- **TensorFlow:** TensorFlow is an open-source machine learning library developed by Google. It is commonly used for building and training deep neural networks for a variety of applications, including computer vision and natural language processing. In sign language

recognition projects, TensorFlow is often used for building and training machine learning models, including deep learning models such as convolutional neural networks (CNNs).

- **Keras:** Keras is a high-level neural networks API written in Python that runs on top of TensorFlow. It is designed to simplify the process of building, training, and deploying deep learning models. In sign language recognition projects, Keras is often used for creating and training deep learning models, as well as evaluating their performance.
- **PIL:** The PIL (Python Imaging Library) module provides basic image processing capabilities that can be used to preprocess images before feeding them into machine learning models.
- **OS module:** The OS module provides a way to interact with the operating system, including functions for file I/O and directory manipulation. This can be useful for loading and saving data to and from disk.
- **Sys module:** The sys module provides access to some system-specific parameters and functions. It can be used to control the Python interpreter and obtain information about the Python environment.

4. Software Design

4.1 Recognition of Hand Gestures:

4.1.1 Data Collection:

Data collection is the process of gathering and recording information or data for a particular purpose. In the context of sign language recognition, data collection involves gathering video or motion capture data of people performing various sign language gestures and sentences.

This data is then used to train machine learning models that can recognize and interpret sign language gestures and sentences. The more data that is collected, the more accurate and robust the machine learning models become, resulting in better sign language recognition.

Data collection is crucial in sign language recognition because sign languages are complex and often involve subtle differences in hand shape, movement, and position. By collecting a large and diverse dataset of sign language gestures, the machine learning models can learn to recognize these subtleties and accurately interpret the intended meaning behind the signs.

Fig. 1 Raw image of an alphabet

4.1.2. Data Pre-Processing:

Data preprocessing is the process of cleaning, transforming, and preparing data for analysis. In the context of sign language recognition, data preprocessing involves preparing the\ motion capture data collected during data collection for use in machine learning models.

Various Filters Used In Image Pre-Processing:

In sign language recognition, image preprocessing techniques involve applying various filters to the images to remove noise, enhance the relevant features, and improve the overall quality of the images. Here are some commonly used filters in image preprocessing for sign language recognition:

- **Grayscale filter** is a common image preprocessing technique that involves converting a color image into a grayscale image. The grayscale filter converts a color image to grayscale by taking the average of the red, green, and blue (RGB) values for each pixel and assigning the same value to each of the three color channels. The resulting grayscale image has a single channel, with each pixel value representing the brightness level of that pixel. The brightness level of each pixel is often calculated using a weighted average of the RGB values, to account for differences in the perceived brightness of different colors.
- **Gaussian blur** filter is a technique used in image preprocessing to smooth an image and reduce the noise present in the image. It works by convolving the input image with a Gaussian function, which is a bell-shaped curve that assigns weights to each pixel based on its distance from the center pixel. The closer a pixel is to the center, the higher the weight assigned to it. The convolution process averages the pixel values within a neighborhood of pixels, resulting in a blur effect that reduces high-frequency details and smoothes out noise.

The amount of blur applied to the image can be controlled by adjusting the standard deviation of the Gaussian function or the size of the kernel used in the convolution.

- **Thresholding** is a common image preprocessing technique used in sign language recognition that involves converting a grayscale image into a binary image by separating the pixels into two categories based on a threshold value. Pixels with brightness values above the threshold value are assigned one value (often white), while pixels with brightness values below the threshold value are assigned another value (often black).
- **Adaptive thresholding** is a technique that adjusts the threshold value locally in different regions of the image. This can be useful in sign language recognition, where lighting conditions and background can vary greatly between different signs and signers. By applying adaptive thresholding, we can adjust the threshold value based on the local properties of the image and improve recognition accuracy.

Overall, grayscale, Gaussian blur, threshold, and adaptive threshold are all useful image processing techniques that can help to improve the quality of sign language data and increase recognition accuracy.

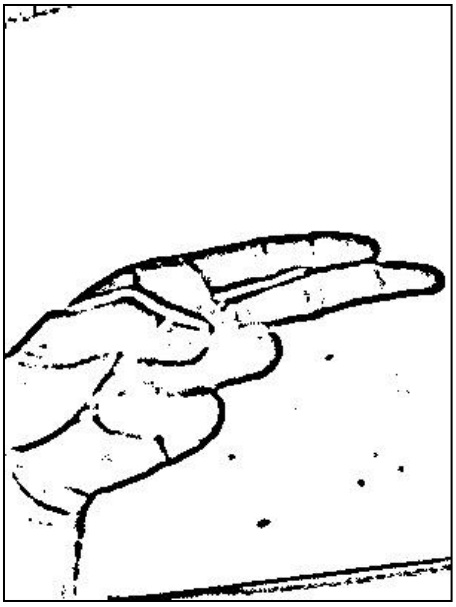


Fig. 2 An alphabet after applying filters (Image preprocessed)

4.2 Conversion Of Sign Language Into Words :

4.2.1 Convolutional Neural Network (CNN):

A Convolutional Neural Network (CNN) is a type of deep learning algorithm designed to recognize patterns in images, videos, and other multi-dimensional data. CNNs are widely used in computer vision tasks such as object detection, image recognition, and segmentation.

The basic idea behind CNNs is to extract relevant features from the input data through convolutional layers. These layers apply a set of filters (also called kernels) to the input data, which convolve (or slide) over the input data and produce a feature map. Each filter learns to detect a specific feature in the input data, such as edges, corners, or textures.

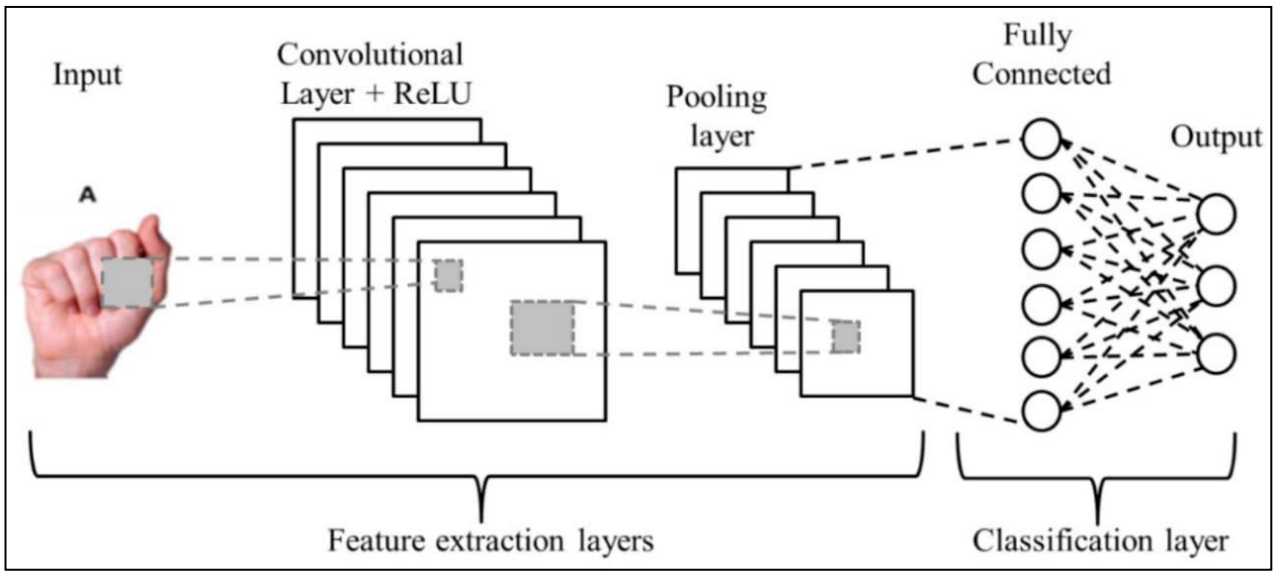


Fig.3 Convolutional Neural Network (CNN)

4.2.2 Sequential Model of Convolutional Neural Network (CNN):

In our project we uses the Sequential model of CNN. A Sequential model of CNN is a type of neural network architecture used for image processing and computer vision tasks. It is a linear

stack of layers where the output of one layer is fed as input to the next layer. This model is particularly useful when the data flows from input to output in a sequential order.

The main advantage of using a sequential model in CNN is that it allows for the automatic feature extraction and learning of hierarchical representations from the input data. This is achieved through the use of convolutional layers, which apply a set of filters to the input image and generate feature maps that highlight important patterns and features in the image. The output of the convolutional layers is then fed into pooling layers, which down sample the feature maps and reduce the dimensionality of the data.

Here are some of the main components of a Sequential model of CNN:

Input layer: This layer is responsible for receiving the input image and preparing it for further processing. The input layer usually takes a 3D tensor representing the image with dimensions (height, width, channels).

Convolutional layer: The convolutional layer is the primary building block of CNNs. It applies a set of filters (also known as kernels or weights) to the input image to generate a set of feature maps. Each filter extracts specific features from the image by performing element-wise multiplication with the input image and summing the results.

Activation layer: The activation layer applies a non-linear activation function to the output of the convolutional layer. This allows the CNN to learn more complex and abstract features from the input image. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh.

Pooling layer: The pooling layer is responsible for downsampling the feature maps generated by the convolutional layer. This helps to reduce the dimensionality of the data and make the model

more efficient. The most common pooling operation is max pooling, where the maximum value in each subregion of the feature map is selected.

Flatten layer: The flatten layer is responsible for flattening the output of the previous layer into a 1D tensor. This prepares the data for input into the fully connected layers.

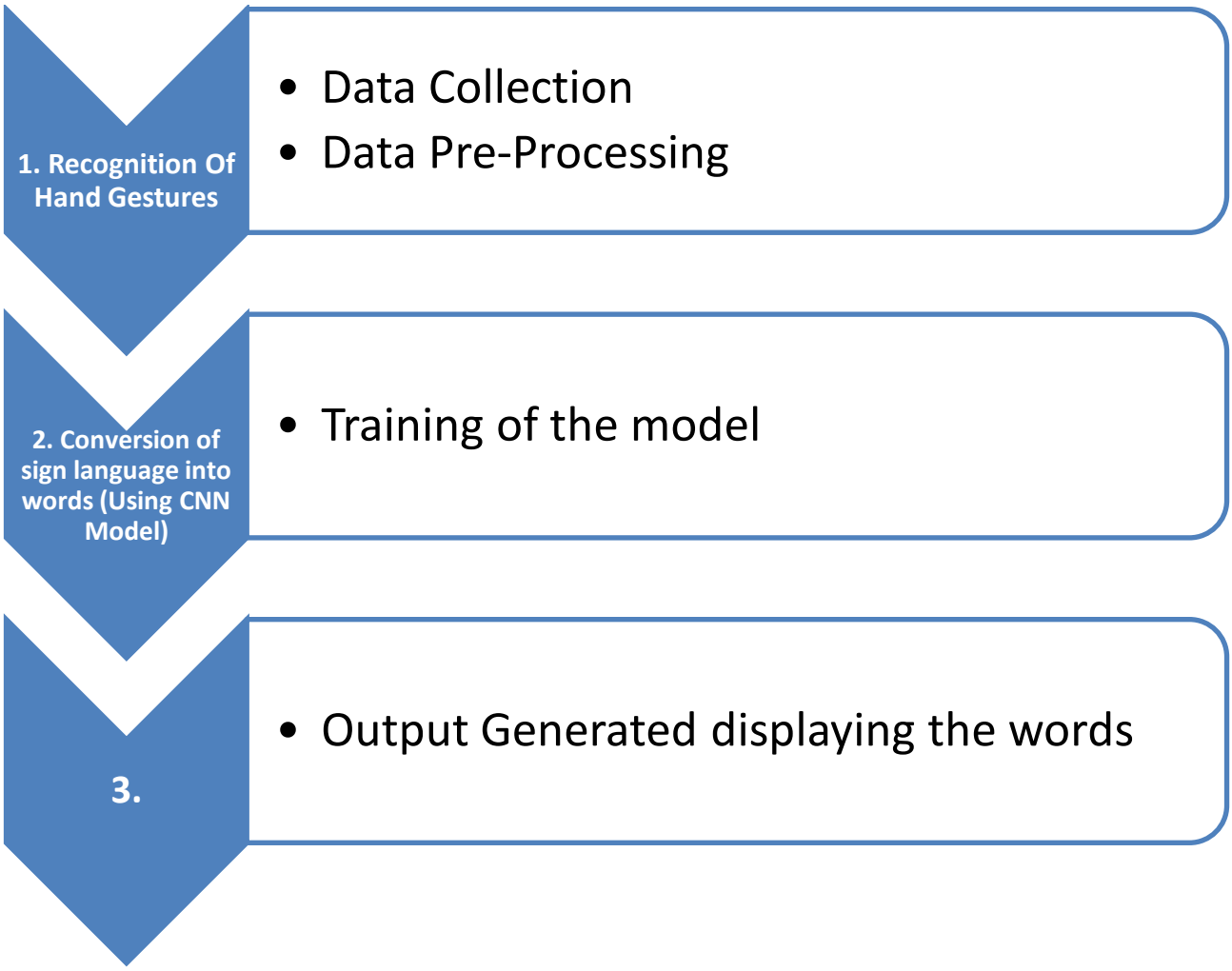
Fully connected layer: The fully connected layer is a traditional neural network layer that connects every neuron in the layer to every neuron in the previous layer. The fully connected layer takes the flattened tensor from the previous layer as input and produces an output tensor representing the probability distribution over the possible classes.

4.3 Conversion to Word:

The project will recognize and interpret sign language gestures and movements made by individuals with hearing or speech impairments and then convert that gesture into words.

4.4 Flowchart for designing the project:

A flowchart is a graphical representation of the steps or activities involved in a project. It helps to visually outline the sequence of steps, decision points, and interactions between various components or stakeholders in the project. Here is the flowchart for our project:



4.5 Gantt Chart

A Gantt chart is a popular project management tool used to represent a project schedule in a visual way. A Gantt chart consists of a horizontal timeline and a set of bars that represent different tasks or activities in the project. Each bar is positioned according to the start and end dates of the task, and its length corresponds to the duration of the task. Dependencies between tasks can also be represented using arrows, connecting one task to another.

ID	Name	Start Date	End Date	Duration	Progress %
1	SIGN LANGUAGE RECOGNITION	Feb 28, 2023	May 15, 2023	55 days	60
2	Getting through the sign language	Feb 28, 2023	Mar 15, 2023	12 days	100
3	Suitable data collection	Mar 16, 2023	Mar 29, 2023	10 days	100
4	Data Pre-Processing	Mar 30, 2023	Apr 07, 2023	7 days	100
5	Training of Pre-Processed data	Apr 07, 2023	Apr 25, 2023	13 days	40
6	Prediction of Pre-Processed Data	Apr 26, 2023	May 08, 2023	9 days	0
7	Testing	May 08, 2023	May 15, 2023	6 days	0

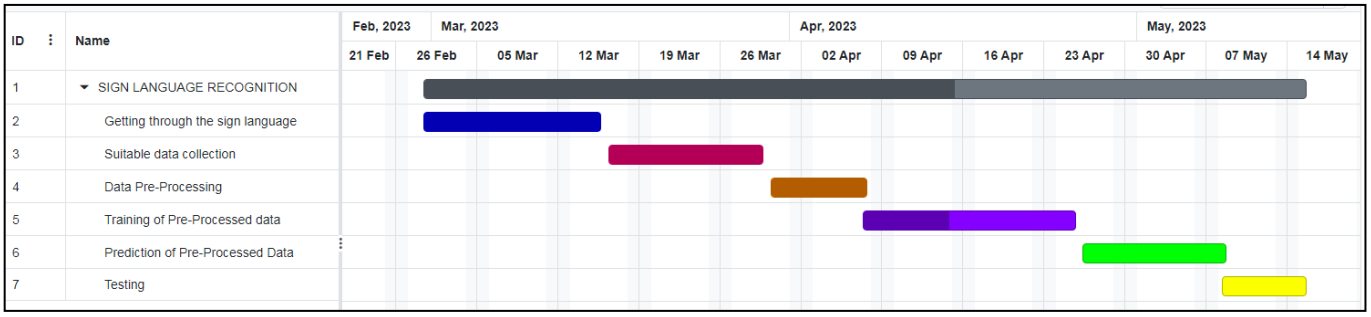


Fig. 4 Gantt chart

5. Coding / Core Module

5.1 Data Collection

```
import cv2
importnumpyas np
importos
import string
# Create the directory structure
ifnotos.path.exists("data"):
    os.makedirs("data")
ifnotos.path.exists("data/train"):
    os.makedirs("data/train")
ifnotos.path.exists("data/test"):
    os.makedirs("data/test")
foriinrange(10):
    ifnotos.path.exists("data/train/" + str(i)):
        os.makedirs("data/train/"+str(i))
    ifnotos.path.exists("data/test/" + str(i)):
        os.makedirs("data/test/"+str(i))

foriinstring.ascii_uppercase:
    ifnotos.path.exists("data/train/" + i):
        os.makedirs("data/train/"+i)
    ifnotos.path.exists("data/test/" + i):
        os.makedirs("data/test/"+i)

# Train or test
mode = 'train'
directory = 'data/'+mode+'/'
minValue = 70

cap = cv2.VideoCapture(0)
interrupt = -1

whileTrue:
    _, frame = cap.read()
    # Simulating mirror image
    frame = cv2.flip(frame, 1)

    # Getting count of existing images
    count = {
            '0': len(os.listdir(directory+"/0")),
            '1': len(os.listdir(directory+"/1")),
            '2': len(os.listdir(directory+"/2")),
            '3': len(os.listdir(directory+"/3")),
            '4': len(os.listdir(directory+"/4")),
```

```
            '5': len(os.listdir(directory+"/5")),
            '6': len(os.listdir(directory+"/6")),
            '7': len(os.listdir(directory+"/7")),
            '8': len(os.listdir(directory+"/8")),
            '9': len(os.listdir(directory+"/9")),
            'a': len(os.listdir(directory+"/A")),
            'b': len(os.listdir(directory+"/B")),
            'c': len(os.listdir(directory+"/C")),
            'd': len(os.listdir(directory+"/D")),
            'e': len(os.listdir(directory+"/E")),
            'f': len(os.listdir(directory+"/F")),
            'g': len(os.listdir(directory+"/G")),
            'h': len(os.listdir(directory+"/H")),
            'i': len(os.listdir(directory+"/I")),
            'j': len(os.listdir(directory+"/J")),
            'k': len(os.listdir(directory+"/K")),
            'l': len(os.listdir(directory+"/L")),
            'm': len(os.listdir(directory+"/M")),
            'n': len(os.listdir(directory+"/N")),
            'o': len(os.listdir(directory+"/O")),
            'p': len(os.listdir(directory+"/P")),
            'q': len(os.listdir(directory+"/Q")),
            'r': len(os.listdir(directory+"/R")),
            's': len(os.listdir(directory+"/S")),
            't': len(os.listdir(directory+"/T")),
            'u': len(os.listdir(directory+"/U")),
            'v': len(os.listdir(directory+"/V")),
            'w': len(os.listdir(directory+"/W")),
            'x': len(os.listdir(directory+"/X")),
            'y': len(os.listdir(directory+"/Y")),
            'z': len(os.listdir(directory+"/Z"))
            }

    # Printing the count in each set to the screen
    # cv2.putText(frame, "MODE : "+mode, (10, 50), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    # cv2.putText(frame, "IMAGE COUNT", (10, ), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "ZERO : "+str(count['0']), (10, 70), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "ONE : "+str(count['1']), (10, 80), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "TWO : "+str(count['2']), (10, 90), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "THREE : "+str(count['3']), (10, 100), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "FOUR : "+str(count['4']), (10, 110), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
```

```
    cv2.putText(frame, "FIVE : "+str(count['5']), (10, 120), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "SIX : "+str(count['6']), (10, 130), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "SEVEN : "+str(count['7']), (10, 140), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "EIGHT : "+str(count['8']), (10, 150), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "NINE : "+str(count['9']), (10, 160), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "a : "+str(count['a']), (10, 170), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "b : "+str(count['b']), (10, 180), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "c : "+str(count['c']), (10, 190), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "d : "+str(count['d']), (10, 200), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "e : "+str(count['e']), (10, 210), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "f : "+str(count['f']), (10, 220), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "g : "+str(count['g']), (10, 230), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "h : "+str(count['h']), (10, 240), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "i : "+str(count['i']), (10, 250), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "k : "+str(count['k']), (10, 260), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "l : "+str(count['l']), (10, 270), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "m : "+str(count['m']), (10, 280), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "n : "+str(count['n']), (10, 290), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "o : "+str(count['o']), (10, 300), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "p : "+str(count['p']), (10, 310), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "q : "+str(count['q']), (10, 320), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "r : "+str(count['r']), (10, 330), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "s : "+str(count['s']), (10, 340), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "t : "+str(count['t']), (10, 350), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
```

```
    cv2.putText(frame, "u : "+str(count['u']), (10, 360), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "v : "+str(count['v']), (10, 370), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "w : "+str(count['w']), (10, 380), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "x : "+str(count['x']), (10, 390), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "y : "+str(count['y']), (10, 400), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    cv2.putText(frame, "z : "+str(count['z']), (10, 410), cv2.FONT_HERSHEY_PLAIN, 1,
(0,255,255), 1)
    # Coordinates of the ROI
    roi = frame[10:410, 220:520]

    cv2.imshow("Frame", frame)
    gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

    blur = cv2.GaussianBlur(gray,(5,5),2)
    # #blur = cv2.bilateralFilter(roi,9,75,75)

    th3 =
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINA
RY_INV,11,2)
    ret, test_image = cv2.threshold(th3, minValue, 255,
cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

    test_image = cv2.resize(test_image, (300,300))
    cv2.imshow("test", test_image)

    interrupt = cv2.waitKey(10)
    if interrupt &0xFF == 27: # esc key
        break
    if interrupt &0xFF == ord('0'):
        cv2.imwrite(directory+'0/'+str(count['0'])+'.jpg', roi)
    if interrupt &0xFF == ord('1'):
        cv2.imwrite(directory+'1/'+str(count['1'])+'.jpg', roi)
    if interrupt &0xFF == ord('2'):
        cv2.imwrite(directory+'2/'+str(count['2'])+'.jpg', roi)
    if interrupt &0xFF == ord('3'):
        cv2.imwrite(directory+'3/'+str(count['3'])+'.jpg', roi)
    if interrupt &0xFF == ord('4'):
        cv2.imwrite(directory+'4/'+str(count['4'])+'.jpg', roi)
    if interrupt &0xFF == ord('5'):
        cv2.imwrite(directory+'5/'+str(count['5'])+'.jpg', roi)
    if interrupt &0xFF == ord('6'):
        cv2.imwrite(directory+'6/'+str(count['6'])+'.jpg', roi)
    if interrupt &0xFF == ord('7'):
        cv2.imwrite(directory+'7/'+str(count['7'])+'.jpg', roi)
```

```
if interrupt &0xFF == ord('8'):
    cv2.imwrite(directory+'8/'+str(count['8'])+'.jpg', roi)
if interrupt &0xFF == ord('9'):
    cv2.imwrite(directory+'9/'+str(count['9'])+'.jpg', roi)
if interrupt &0xFF == ord('a'):
    cv2.imwrite(directory+'A/'+str(count['a'])+'.jpg', roi)
if interrupt &0xFF == ord('b'):
    cv2.imwrite(directory+'B/'+str(count['b'])+'.jpg', roi)
if interrupt &0xFF == ord('c'):
    cv2.imwrite(directory+'C/'+str(count['c'])+'.jpg', roi)
if interrupt &0xFF == ord('d'):
    cv2.imwrite(directory+'D/'+str(count['d'])+'.jpg', roi)
if interrupt &0xFF == ord('e'):
    cv2.imwrite(directory+'E/'+str(count['e'])+'.jpg', roi)
if interrupt &0xFF == ord('f'):
    cv2.imwrite(directory+'F/'+str(count['f'])+'.jpg', roi)
if interrupt &0xFF == ord('g'):
    cv2.imwrite(directory+'G/'+str(count['g'])+'.jpg', roi)
if interrupt &0xFF == ord('h'):
    cv2.imwrite(directory+'H/'+str(count['h'])+'.jpg', roi)
if interrupt &0xFF == ord('i'):
    cv2.imwrite(directory+'I/'+str(count['i'])+'.jpg', roi)
if interrupt &0xFF == ord('j'):
    cv2.imwrite(directory+'J/'+str(count['j'])+'.jpg', roi)
if interrupt &0xFF == ord('k'):
    cv2.imwrite(directory+'K/'+str(count['k'])+'.jpg', roi)
if interrupt &0xFF == ord('l'):
    cv2.imwrite(directory+'L/'+str(count['l'])+'.jpg', roi)
if interrupt &0xFF == ord('m'):
    cv2.imwrite(directory+'M/'+str(count['m'])+'.jpg', roi)
if interrupt &0xFF == ord('n'):
    cv2.imwrite(directory+'N/'+str(count['n'])+'.jpg', roi)
if interrupt &0xFF == ord('o'):
    cv2.imwrite(directory+'O/'+str(count['o'])+'.jpg', roi)
if interrupt &0xFF == ord('p'):
    cv2.imwrite(directory+'P/'+str(count['p'])+'.jpg', roi)
if interrupt &0xFF == ord('q'):
    cv2.imwrite(directory+'Q/'+str(count['q'])+'.jpg', roi)
if interrupt &0xFF == ord('r'):
    cv2.imwrite(directory+'R/'+str(count['r'])+'.jpg', roi)
if interrupt &0xFF == ord('s'):
    cv2.imwrite(directory+'S/'+str(count['s'])+'.jpg', roi)
if interrupt &0xFF == ord('t'):
    cv2.imwrite(directory+'T/'+str(count['t'])+'.jpg', roi)
if interrupt &0xFF == ord('u'):
    cv2.imwrite(directory+'U/'+str(count['u'])+'.jpg', roi)
if interrupt &0xFF == ord('v'):
    cv2.imwrite(directory+'V/'+str(count['v'])+'.jpg', roi)
if interrupt &0xFF == ord('w'):
```

```
        cv2.imwrite(directory+'W/'+str(count['w'])+'.jpg', roi)
    if interrupt &0xFF == ord('x'):
        cv2.imwrite(directory+'X/'+str(count['x'])+'.jpg', roi)
    if interrupt &0xFF == ord('y'):
        cv2.imwrite(directory+'Y/'+str(count['y'])+'.jpg', roi)
    if interrupt &0xFF == ord('z'):
        cv2.imwrite(directory+'Z/'+str(count['z'])+'.jpg', roi)

cap.release()
cv2.destroyAllWindows()
```

5.2 Data Pre-Processing

```
importnumpyas np
import cv2
importos
import csv
from PIL import Image, ImageTk
fromimage_processingimportfunc
ifnotos.path.exists("data2"):
    os.makedirs("data2")
ifnotos.path.exists("data2/train"):
    os.makedirs("data2/train")
ifnotos.path.exists("data2/test"):
    os.makedirs("data2/test")
path="data/train"
path1 = "data2"

label=0
var = 0
c1 = 0
c2 = 0

for (dirpath,dirnames,filenames) inos.walk(path):
    fordirnameindirnames:
        for(direcpath,direcnames,files) inos.walk(path+"/"+dirname):
            # print('direcnames = ',direcnames)
            ifnotos.path.exists(path1+"/train/"+dirname):
                os.makedirs(path1+"/train/"+dirname)
            ifnotos.path.exists(path1+"/test/"+dirname):
                os.makedirs(path1+"/test/"+dirname)
            num=0.80*len(files)
            i=0

            for file in files:
                var+=1
                actual_path=path+"/"+dirname+"/"+file
```

```
            actual_path1=path1+"/"+"train/"+dirname+"/"+file
            actual_path2=path1+"/"+"test/"+dirname+"/"+file
            img = cv2.imread(actual_path, 0)
            bw_image = func(actual_path)
            ifi<num:
               c1 += 1
               cv2.imwrite(actual_path1 ,bw_image)
            else:
               c2 += 1
               cv2.imwrite(actual_path2 ,bw_image)

            i=i+1

      label=label+1
print(var)
print(c1)
print(c2)
```

5.2.1 Image Pre-Processing

```
importnumpyas np
import cv2
minValue = 70
deffunc(path):
   frame = cv2.imread(path)

   gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
   blur = cv2.GaussianBlur(gray,(5,5),2)

   th3 =
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINA
RY_INV,11,2)
   ret, res = cv2.threshold(th3, minValue, 255,
cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
   return res
```

6. Performance of the project developed

6.1 Data Collection

Data collection is a critical part of our project as it allows our machine learning algorithm to learn and recognize different hand gestures with greater accuracy. The more data we collect, the better our algorithm will be at recognizing different hand gestures and converting them into language.

When collecting data for our project, it is important to ensure that we capture a wide variety of hand gestures and in different lighting conditions. This will help to make our algorithm more robust and adaptable to different scenarios.

6.2 Data Pre-Processing

Data pre-processing is a crucial step in our hand gesture recognition project as it can help to improve the accuracy and performance of our machine learning model. Here are some ways in which the preprocessing techniques we mentioned - Gaussian blur and adaptive thresholding - can be helpful:

- **Image Details:** By applying Gaussian blur, we can reduce the details in the image, making it easier for our model to identify the important features of the hand gesture.
- **Feature extraction:** Adaptive thresholding can help to extract the key features of the hand gesture, such as the outline or contour. This can help our model to identify the gesture more accurately.

8. Output Screens

8.1 Data Collection

Fig. 5.1 Digit 1

Fig. 5.2 Alphabet B

Fig. 5.3 Alphabet R

Fig. 5.4 Alphabet Y

8.2 Data Pre-Processing

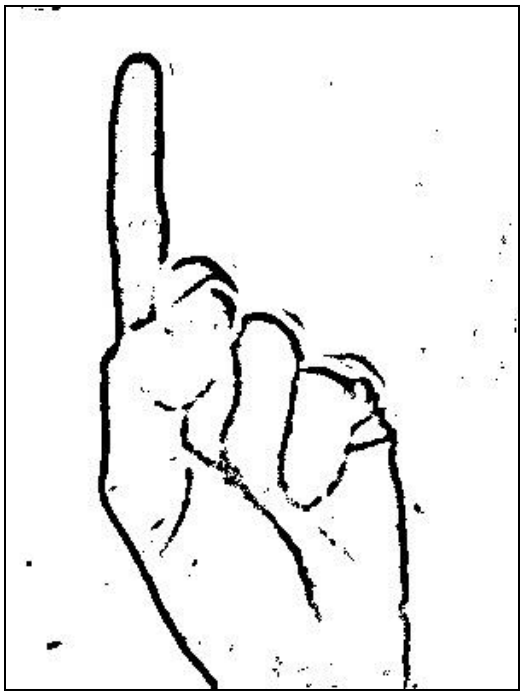


Fig 5.5 Digit 1

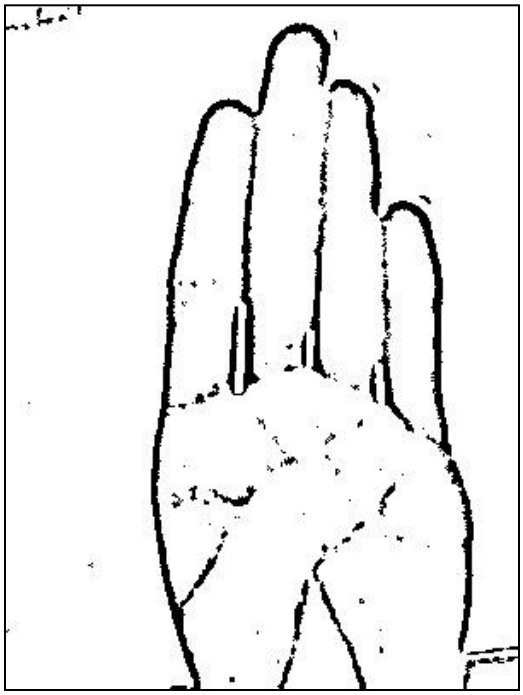


Fig. 5.6 Alphabet B

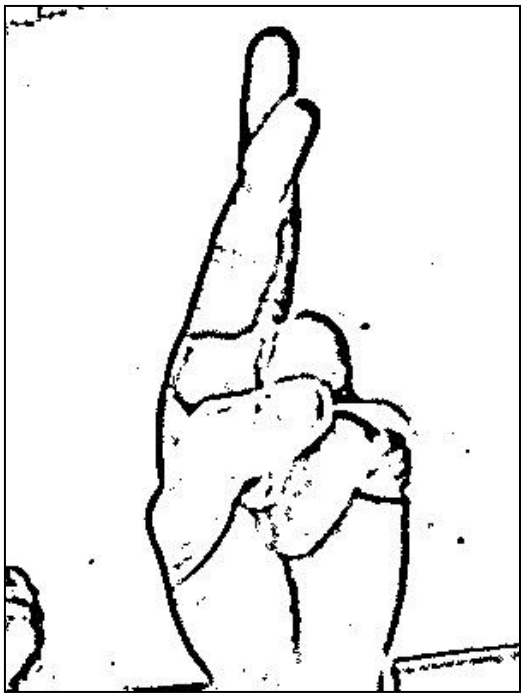


Fig. 5.7 Alphabet R

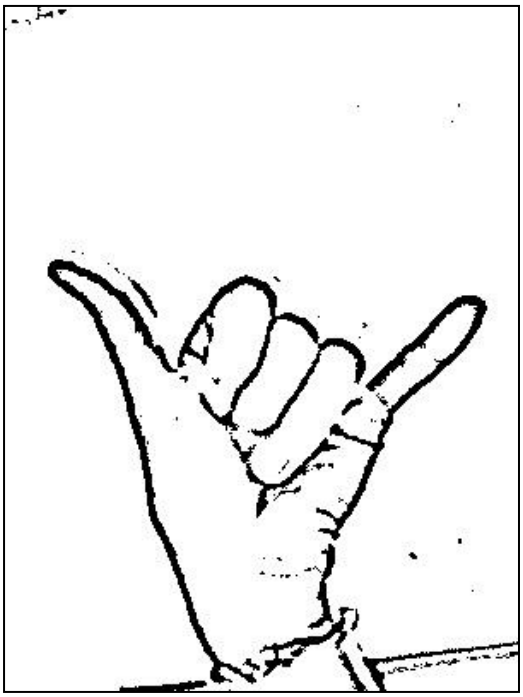


Fig. 5.8 Alphabet Y

9. References

[1]Buehler, T., Everingham, M., & Zisserman, A. (2009). Recognition and synthesis of hand-squeezes in sign language videos. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 859-866).

[2] Yao, T., Li, Y., Huang, Y., Li, Z., & Jiang, T. (2018). Sign language recognition using 3D convolutional neural networks. IEEE Access, 6, 14429-14438.

[3] Garcia-Hernandez, N., Camarena-Ibarrola, A., & Carrasco-Ochoa, J. A. (2017). Sign language recognition based on fuzzy inference systems and image processing. IEEE Transactions on Human-Machine Systems, 47(2), 174-183.

[4] Zhou, L., & Wong, K. Y. K. (2020). Sign language recognition using deep learning: A survey. IEEE Transactions on Neural Systems and Rehabilitation Engineering, 28(1), 4-16.

[5] Lee, D., & Kim, K. (2019). Sign language recognition based on deep learning with adaptive hand gesture segmentation. IEEE Access, 7, 12781-12790.