THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Hand gesture Recognition using CV

Submitted By:

Prem Jadwani (102103215)
Krishna Khanduja(102153007)

Submitted To:

Mr. Arun Pundir

July 2023 – December 2023

INTRODUCTION

Sign language serves as a fundamental mode of communication for the deaf and hard-of-hearing community, offering a rich and expressive means of conveying thoughts, emotions, and ideas. Among the various sign languages, American Sign Language (ASL) holds particular significance, providing a unique linguistic system that enables deaf individuals to communicate with each other and the broader society. Despite the prevalence and importance of ASL, a communication gap persists between the deaf community and the hearing population.

Advancements in technology, particularly in the fields of machine learning (ML) and computer vision, have presented unprecedented opportunities to bridge this communication divide. Recognizing hand signs in ASL through automated systems has emerged as a promising avenue for fostering inclusivity and accessibility. This project seeks to contribute to this burgeoning field by developing a machine learning model specifically designed for hand sign detection in ASL.

The primary motivation behind this endeavor lies in the potential transformative impact such a system could have on the lives of individuals with hearing impairments. By creating an accurate and efficient hand sign detection model, we aim to empower the deaf community with enhanced communication tools, enabling them to participate more fully in various aspects of daily life, from education and employment to social interactions.

LITERATURE SURVEY

Previous Approaches in Sign Language Recognition:

A comprehensive review of the existing literature reveals a spectrum of approaches to sign language recognition. Traditional methods often relied on handcrafted features and rule-based systems, while more recent studies leverage the power of deep learning to automatically extract intricate patterns and representations from visual data.

Deep Learning for Hand Gesture Recognition:

The rise of deep learning models, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), has significantly advanced the capabilities of hand gesture recognition systems. These models excel in learning complex spatial and temporal relationships, crucial for accurately interpreting the dynamic nature of ASL.

Datasets and Annotations:

The choice of datasets plays a pivotal role in the training and evaluation of hand sign detection models. Investigating existing datasets curated for ASL, such as the American Sign Language Alphabet Dataset, aids in understanding the diversity of gestures and challenges associated with real-world applications.

Challenges in ASL Recognition:

The multifaceted nature of ASL presents challenges, including variations in hand shapes, dynamic movements, and the influence of facial expressions. Previous research has explored techniques to address these challenges, such as incorporating temporal information, considering hand dynamics, and developing robust solutions for diverse lighting conditions.

Real-world Applications and Impact:

Beyond academic exploration, the practical applications of hand sign detection extend to realms such as assistive technologies, human-computer interaction, and the creation of inclusive environments. Understanding the tangible impact of sign language recognition systems informs the design and development process, emphasizing the importance of creating solutions that resonate with real-world needs.

State-of-the-Art Models and Techniques:

Staying abreast of the latest advancements in models and techniques, including Transformer-based architectures and attention mechanisms, ensures that our project aligns with the cutting edge of hand sign detection technology. This knowledge informs our approach and positions the project within the broader landscape of current research.

PYTHON LIBRARY USED

NumPy:

NumPy, short for Numerical Python, is a fundamental library in Python for numerical computing. It introduces support for large, multi-dimensional arrays and matrices, along with an extensive collection of high-level mathematical functions to operate on these arrays. NumPy's key strength lies in its ability to perform efficient and vectorized operations, making it indispensable for scientific computing, data analysis, and machine learning. It forms the foundation for many other libraries in the Python ecosystem, enabling seamless integration and interoperability with various data science tools.

OpenCV (Open Source Computer Vision):

OpenCV is a versatile open-source computer vision and image processing library that provides a comprehensive suite of tools for analyzing and manipulating visual data. Widely used in computer vision applications, OpenCV includes functionalities for image and video processing, feature detection, object tracking, and machine learning. Its broad range of algorithms makes it a go-to solution for tasks like image recognition, augmented reality, and robotics.

TensorFlow:

TensorFlow, developed by Google, is a powerful open-source machine learning framework designed to simplify the development and deployment of machine learning models, especially neural networks. TensorFlow offers a flexible and scalable platform for building various types of machine learning models, including deep learning. It supports efficient training and deployment across diverse platforms, enabling researchers and developers to tackle complex machine learning tasks effectively. TensorFlow has gained widespread adoption in both academia and industry, contributing to advancements in artificial intelligence and machine learning applications.

CODE EXPLANATION

DATA COLLECTION CODE-:

```
import cv2
from cvzone.HandTrackingModule import HandDetector
#hand detection library
import numpy as np
import math
import time


cap=cv2.VideoCapture(0)
# to capture web cam and 0 is default value of web cam
detector=HandDetector(maxHands=1) #detect total of 1 hand only to keep it simple

offset=20
imageSize = 300

folder="Data/C" #images when pressed s will be saved here
counter=0

while True:
    success, img=cap.read()
    hands, img=detector.findHands(img)#it will detect hand when web cam opens
    if hands:
        hand=hands[0]#means only one hand , if max hands was 2 then hands[1] means second
hand
        x,y,w,h=hand['bbox']#gives dimensions of hand

        #create image by ourselves with white background and then oervlay the image on that white
bacground and then resize and centre it so thats it fits the white box
        #created from matrix
        imageWhite = np.ones((imageSize,imageSize,3), np.uint8)*255
        #square matrix of 300 by 300 and 3 means colored image i.e r,g,b
        #also give range of values and in our case its from 0 to 255 so iuse unsigned integer
        imgCrop = img[y-offset:y+h+offset , x-offset:x+w+offset]#gives spaces for hand images

        imgCropShape=imgCrop.shape
        #image resiszing of height so that it fits the white background
        #if height>width ,then make hight =300
        # if width>height ,then stretch width to 300

        ascpetRatio = h/w


        #for height
        if ascpetRatio>1: #means height is bigger
            k=imageSize/h
            wCal=math.ceil(k*w)
            imgResize=cv2.resize(imgCrop,(wCal,imageSize))
            imageResizeShape=imgResize.shape
            #now to centre the image
            wGap=math.ceil((imageSize-wCal)/2)
            imageWhite[:,wGap:wCal+wGap] = imgResize  # this is gonna lay our image on white
background
            # 0-starting point of height , imgCropShape[0]-end point of height and same for width


        #for width
        else:
            k = imageSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop, (imageSize, hCal))
            imageResizeShape = imgResize.shape
            # now to centre the image
            hGap = math.ceil((imageSize - hCal) / 2)
            imageWhite[hGap:hCal + hGap,:] = imgResize




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


    cv2.imshow("Image" , img) #open webcam and title of tab will be image
    key = cv2.waitKey(1) #wait of 1 milisecond
    if key==ord("s"):
        counter+=1
        #to tell how many images are saved
        #above will trigger only when s key is pressed
        cv2.imwrite(f'{folder}/Image_{time.time()}.jpg',imageWhite)
        #it means white images will be saved on folder with name Iamge_1,2,3... as time .time will
give different values everytime
        print(counter)
```

EXPLANATION:

Library Import:

The code begins by importing necessary libraries.CV2 is used for computer vision tasks.HandDetector from cvzone is employed for hand tracking.Numpy is utilized for numerical operations.Math provides mathematical functions.Time is included for handling timestamps.

Initialization:

The VideoCapture object (cap) is initialized to access the webcam (device index 0).
A HandDetector object (detector) is created with the parameter maxHands set to 1, indicating that it will detect only one hand.

Variable Initialization:

offset defines the space around the detected hand for cropping.
imageSize sets the dimensions for the white background image.
folder specifies the directory where the images will be saved.
counter keeps track of the number of images saved.

Main Loop:

The code enters a continuous loop to capture video frames from the webcam using cap.read().The detector.findHands(img) function detects hands in the frame, and the results are stored in the hands variable.
Hand Detection and Cropping:

If hands are detected (if hands:), the bounding box coordinates of the detected hand are extracted using hand['bbox'].
The region of interest (ROI) around the hand is cropped from the original frame.
White Background Image Creation:

An all-white background image (imageWhite) is created using NumPy, ensuring a 3-channel (RGB) image.

Resizing and Centering:

The cropped hand is resized and centered on the white background.
The aspect ratio is calculated to adjust the height or width based on the larger dimension.
The resized hand is overlaid onto the white background, creating the final image.

Displaying Images:

The code displays both the cropped hand (imgCrop) and the resulting image with a white background (imageWhite) in separate windows.
User Interaction - Saving Images:

The original video frame is displayed in a window titled "Image."
The user can save the resulting images by pressing the 's' key.
Each saved image is timestamped using time.time() and stored in the specified folder with a unique filename.
The counter is incremented to keep track of the number of saved images.

This code essentially captures webcam frames, detects and crops a hand, overlays it onto a white background, and allows the user to save the resulting images with a timestamp. The saved images are stored in a specified folder, and the counter keeps track of the total number of saved images.

TEST CODE:

```
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
#classifier module
#hand detection library
import numpy as np
import math
import time


cap=cv2.VideoCapture(0)
# to capture web cam and 0 is default value of web cam
detector=HandDetector(maxHands=1) #detect total of 1 hand only to keep it simple
classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")

offset=20
imageSize = 300

folder="Data/C" #images when pressed s will be saved here
counter=0
labels = ["A", "B", "C"]

while True:
    success, img=cap.read()
    imgCopy = img.copy()
    hands, img=detector.findHands(img)#it will detect hand when web cam opens
    if hands:
        hand=hands[0]#means only one hand , if max hands was 2 then hands[1] means second hand
        x,y,w,h=hand['bbox']#gives dimensions of hand

        #create image by ourselves with white background and then oervlay the image on that white
bacground and then resize and centre it so thats it fits the white box
        #created from matrix
        imageWhite = np.ones((imageSize,imageSize,3), np.uint8)*255
        #square matrix of 300 by 300 and 3 means colored image i.e r,g,b
        #also give range of values and in our case its from 0 to 255 so iuse unsigned integer
        imgCrop = img[y-offset:y+h+offset , x-offset:x+w+offset]#gives spaces for hand images

        imgCropShape=imgCrop.shape


        #image resiszing of height so that it fits the white background
        #if height>width ,then make hight =300
        # if width>height ,then stretch width to 300

        ascpetRatio = h/w


        #for height
        if ascpetRatio>1: #means height is bigger
            k=imageSize/h
            wCal=math.ceil(k*w)
            imgResize=cv2.resize(imgCrop,(wCal,imageSize))
            imageResizeShape=imgResize.shape
            #now to centre the image
            wGap=math.ceil((imageSize-wCal)/2)
            imageWhite[:,wGap:wCal+wGap] = imgResize  # this is gonna lay our image on white
background
            # 0-starting point of height , imgCropShape[0]-end point of height and same for width
            predicition, index=classifier.getPrediction(imageWhite, draw=False)#get predicition gives two
values one predicition and one index
            print(predicition , index)



        #for width
        else:
            k = imageSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop, (imageSize, hCal))
            imageResizeShape = imgResize.shape
            # now to centre the image
            hGap = math.ceil((imageSize - hCal) / 2)
            imageWhite[hGap:hCal + hGap,:] = imgResize
            predicition, index = classifier.getPrediction(imageWhite, draw=False)

        cv2.rectangle(imgCopy, (x - offset, y - offset-50), (x - offset+90 , y - offset-50+50),(255,0,255),
cv2.FILLED)
        cv2.putText(imgCopy, labels[index], (x,y-25), cv2.FONT_HERSHEY_COMPLEX, 1.7, (255, 255,
255), 2)
```

```
        cv2.rectangle(imgCopy ,(x-offset,y-offset), (x+w+offset,y+h+offset), (255,0,255),4)

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


    cv2.imshow("Image" , imgCopy) #open webcam and title of tab will be image
    cv2.waitKey(1) #wait of 1 milisecond
```

EXPLANATION:

Library Import:

The code imports necessary libraries, including OpenCV for computer vision, HandDetector for hand tracking from the cvzone library, and a Classifier for image classification.

Initialization:

The code initializes a video capture object (cap) to access the webcam (device index 0).
HandDetector is initialized to detect a maximum of one hand.
A Classifier is initialized with a pre-trained Keras model and label file.

Variable Initialization:

Variables like offset, imageSize, and folder are defined for image processing.
counter keeps track of the number of images saved.
labels is a list containing labels for the image classifications.

Main Loop:

The code enters a continuous loop to capture video frames from the webcam.
Hand Detection and Cropping:

If hands are detected, the bounding box coordinates of the detected hand are extracted.

The region of interest (ROI) around the hand is cropped from the original frame.

Image Classification:

The code resizes and centers the cropped hand on a white background.
The getPrediction method from the Classifier is used to obtain the prediction and index of the predicted class.

Visualization:

The code visualizes the classification results by drawing rectangles, labels, and other annotations on the image.

Displaying Images:

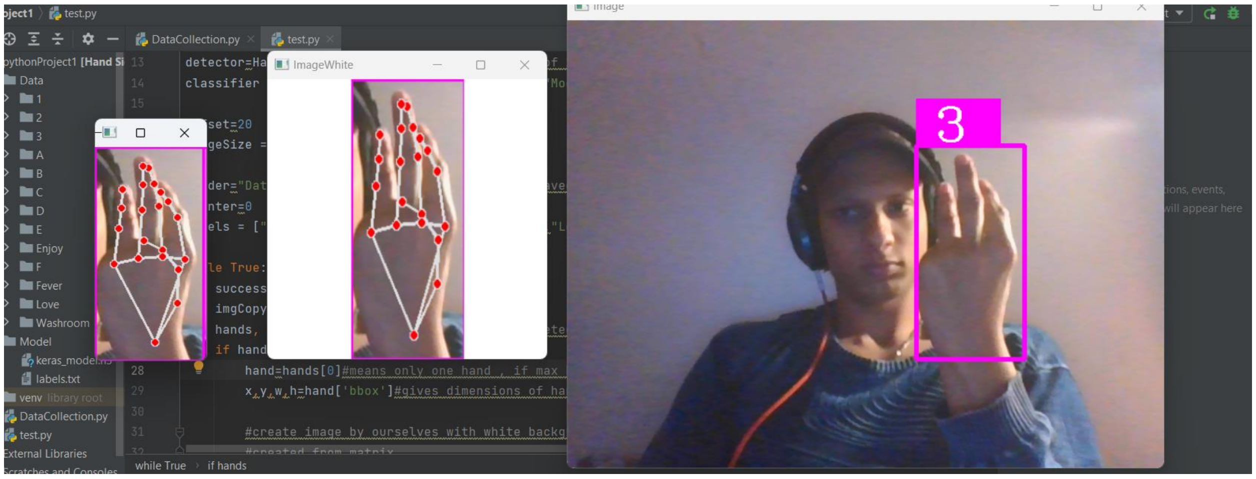
Both the cropped hand and the resulting image with annotations are displayed in separate windows.

User Interaction - Saving Images:

The original video frame is displayed in a window titled "Image."
The loop waits for a key press, and the user can save the resulting images with a timestamp in the specified folder.

This code extends the previous hand detection project by adding an image classification component. It captures webcam frames, detects and classifies a hand gesture, and provides visual feedback on the image with annotations. The classification labels are specified in the labels list, and the resulting images can be saved with a timestamp in the specified folder.

WORKING CODE SNAPSHOTS



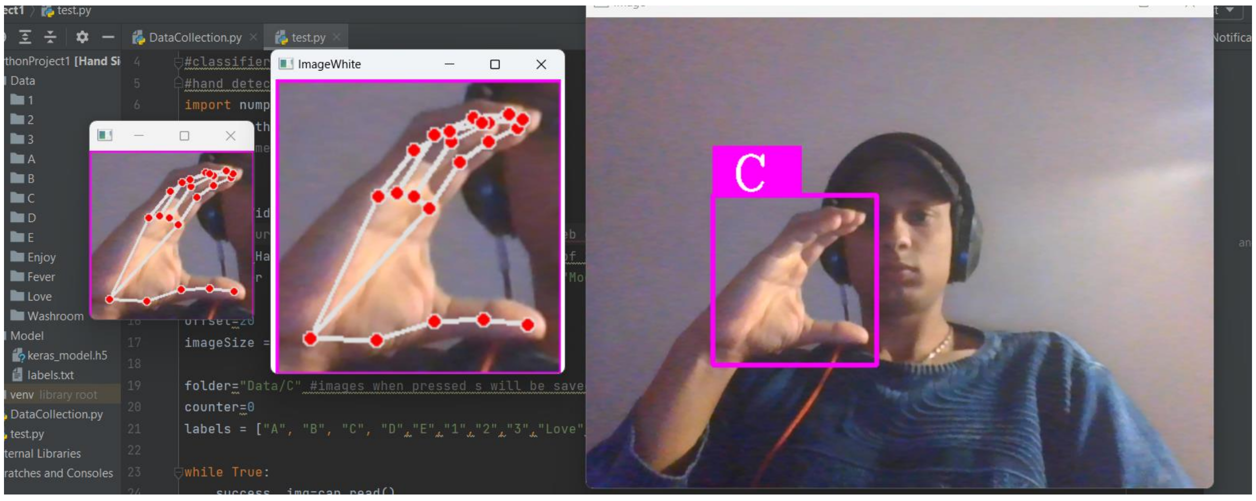
```
detector=Ha
classifier

set=20
geSize =

der="Dat
nter=0
els = ["

le True:
 success
 imgCopy
 hands,
 if hand
     hand=hands[0]#means only one hand , if max
     x,y,w,h=hand['bbox']#gives dimensions of ha

     #create image by ourselves with white backg
```

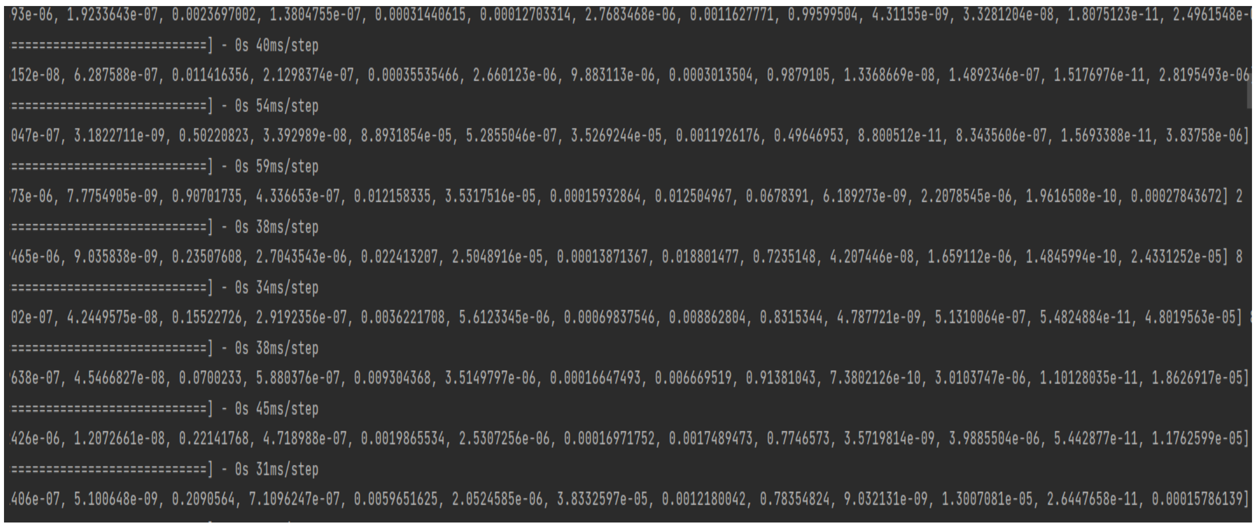


```
#classifier
#hand detec
import nump

imageSize =

folder="Data/C" #images when pressed s will be save
counter=0
labels = ["A", "B", "C", "D","E","1","2","3","Love"

while True:
```



```
93e-06, 1.9233643e-07, 0.0023697002, 1.3804755e-07, 0.00031440615, 0.00012703314, 2.7683468e-06, 0.0011627771, 0.99599504, 4.31155e-09, 3.3281204e-08, 1.8075123e-11, 2.4961548e-
============================] - 0s 40ms/step
152e-08, 6.287588e-07, 0.011416356, 2.1298374e-07, 0.00035535466, 2.660123e-06, 9.883113e-06, 0.0003013504, 0.9879105, 1.3368669e-08, 1.4892346e-07, 1.5176976e-11, 2.8195493e-06
============================] - 0s 54ms/step
047e-07, 3.1822711e-09, 0.50220823, 3.392989e-08, 8.8931854e-05, 5.2855046e-07, 3.5269244e-05, 0.0011926176, 0.49646953, 8.800512e-11, 8.3435606e-07, 1.5693388e-11, 3.83758e-06]
============================] - 0s 59ms/step
73e-06, 7.7754905e-09, 0.90701735, 4.336653e-07, 0.012158335, 3.5317516e-05, 0.00015932864, 0.012504967, 0.0678391, 6.189273e-09, 2.2078545e-06, 1.9616508e-10, 0.00027843672] 2
============================] - 0s 38ms/step
465e-06, 9.035838e-09, 0.23507608, 2.7043543e-06, 0.022413207, 2.5048916e-05, 0.00013871367, 0.018801477, 0.7235148, 4.207446e-08, 1.659112e-06, 1.4845994e-10, 2.4331252e-05] 8
============================] - 0s 34ms/step
02e-07, 4.2449575e-08, 0.15522726, 2.9192356e-07, 0.0036221708, 5.6123345e-06, 0.00069837546, 0.008862804, 0.8315344, 4.787721e-09, 5.1310064e-07, 5.4824884e-11, 4.8019563e-05]
============================] - 0s 38ms/step
638e-07, 4.5466827e-08, 0.0700233, 5.880376e-07, 0.009304368, 3.5149797e-06, 0.00016647493, 0.006669519, 0.91381043, 7.3802126e-10, 3.0103747e-06, 1.10128035e-11, 1.8626917e-05]
============================] - 0s 45ms/step
426e-06, 1.2072661e-08, 0.22141768, 4.718988e-07, 0.0019865534, 2.5307256e-06, 0.00016971752, 0.0017489473, 0.7746573, 3.5719814e-09, 3.9885504e-06, 5.442877e-11, 1.1762599e-05]
============================] - 0s 31ms/step
406e-07, 5.100648e-09, 0.2090564, 7.1096247e-07, 0.0059651625, 2.0524585e-06, 3.8332597e-05, 0.0012180042, 0.78354824, 9.032131e-09, 1.3007081e-05, 2.6447658e-11, 0.00015786139]
```

CONCLUSION

In conclusion, the project aims to bridge the communication gap between the deaf and hearing communities by developing a machine learning model for hand sign detection in American Sign Language (ASL). The literature survey highlights the evolution of sign language recognition techniques, emphasizing the shift from traditional rule-based systems to the power of deep learning, particularly with CNNs and RNNs. Challenges such as variations in hand shapes and dynamic movements in ASL are acknowledged, with a commitment to exploring state-of-the-art models and techniques, including Transformer-based architectures. The Python libraries chosen—NumPy for numerical computing, OpenCV for computer vision, and TensorFlow for machine learning—underscore the project's foundation in robust tools for scientific computing and machine learning applications. Ultimately, the project aspires to contribute to the creation of inclusive environments and assistive technologies, empowering the deaf community in various aspects of daily life.