SIGN LANGUAGE DETECTION USING DEEP LEARNING

HIGHER DIPLOMA IN SOFTWARE ENGINEERING

MACHINE LEARNING - COURSEWORK 2

2023.2F

SUBMITTED BY

M.S.F. Shazna - KUHDSE232F003

W.U.Manchanayaka - KUHDSE232F034

D.M.T.R.Dissanayake -KUHDSE232F040

School of Computing and Engineering

National Institute of Business Management

Kurunegala

Title

Title of the project	-	Sign Language Detection using Deep Learning
Authors	-	M.S.F. Shazna - KUHDSE232F003 W.U.Manchanayaka - KUHDSE232F034 D.M.T.R.Dissanayake -KUHDSE232F040
Name of the Program	-	Higher National Diploma in Software Engineering
Name of the Supervisor	-	Mr. Herath
Name of the Institution	-	School of Computing and Engineering, National Institute of Business Management
Date of Declaration	-	5th December 2023

The Coursework is submitted in partial fulfilment of the requirement of the Higher National Diploma in Software Engineering of National Institute of Business Management.

Declaration

"I certify that this Coursework does not incorporate without acknowledgement, any material previously submitted for a Higher National Diploma in any institution and to the best of my knowledge and belief ,it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my project report, if accepted, to be made available for photocopying and for interlibrary loans, and for the title and summary to be made available to outside organizations".

……..

Signature: Date: 16.12.2023

Abstract

The project focuses on developing a real-time sign language interpretation system to enhance communication accessibility for the deaf and hard-of-hearing community. Leveraging technologies like TensorFlow, OpenCV, and CVZone, the system detects and tracks hand gestures through a webcam. The captured images are processed to create a dataset for training a Convolutional Neural Network (CNN) using Teachable Machine. The trained model interprets American Sign Language (ASL) gestures corresponding to letters A-Z. In the testing phase, the system accurately recognizes gestures, displaying the interpreted letters on-screen. The project's significance lies in its potential to bridge communication gaps, providing a practical solution for individuals who primarily use sign language.

Table of Contents

1. INTRODUCTION

1.1. The Scenario : Background

Envision a scenario where a groundbreaking application is meticulously designed to support individuals who rely on sign language for communication. The primary objective is to establish a trustworthy and precise system adept at interpreting and transcribing sign language gestures into written text. This innovative project holds significant promise in bridging communication disparities and enriching accessibility for the deaf and hard-of-hearing community.

1.2. Objective of the Coursework

The objective of this project is to develop an advanced deep learning model for recognizing hand gestures and interpreting sign language. This project specifically concentrates on implementing a Convolutional Neural Network (CNN) to identify and categorize gestures from American Sign Language (ASL), covering the complete alphabet from A to Z.

1.3. Requirements needed

- A set of images trained and labeled accordingly in the training phase
- Webcam or Camera accessibility
- Python latest version(python 3.11)
- Python libraries;
    - Tensorflow(Open-Source Machine Learning Library)
    - Numpy(Numerical Python)
    - Open CV-Python(Computer vision library)
    - CVZone(Enhances OpenCV functionalities)
    - Mediapipe(ML library for hand tracking)

2. METHODOLOGY

Here two programs should be written, one for Training the dataset and one for testing the final program and get the output. A step-by-step process is given along with relevant codes.

I. Training the dataset (Training.py)

The "Training.py" script is responsible for training the dataset to create a model capable of recognizing American Sign Language (ASL) gestures. It captures images from the webcam, detects hands using the HandTrackingModule, and crops the region of interest . The images are then processed, resized, and stored in a specified folder. This process is repeated each time the user presses the "**a**" key, incrementally building a dataset for each gesture. The dataset is later used to train a deep learning model, enhancing the system's ability to accurately interpret and translate ASL gestures into written text.

2.1. Installation of Required Libraries

If the required libraries are not already available, install it on your device.

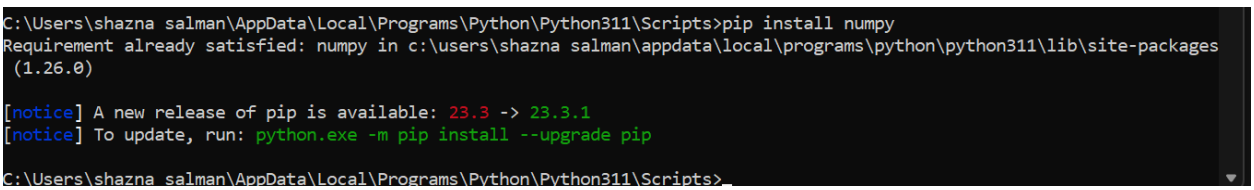
The following are the commands required to install the required libraries;

- pip install tensorflow
- pip install numpy
- pip install opencv-python
- pip install cvzone
- pip install mediapipe

Here is a brief explanation of the functions provided by the libraries.

- **TensorFlow :** TensorFlow is a free and open-source library for machine learning. It's widely used to create and deploy machine learning models, making it accessible for various applications like image recognition and natural language processing.

- **NumPy (Numerical Python):** NumPy is a powerful Python library for numerical computing. It supports large arrays and matrices, along with mathematical functions, making it essential for scientific computing.



- **OpenCV-Python:** OpenCV-Python is a Python binding for OpenCV, a versatile computer vision and machine learning library. It's used for tasks like image processing and object detection.

```
C:\Users\shazna salman\AppData\Local\Programs\Python\Python311\Scripts>pip install opencv-python
Requirement already satisfied: opencv-python in c:\users\shazna salman\appdata\local\programs\python\python311\lib\site-
packages (4.8.1.78)
Requirement already satisfied: numpy>=1.21.2 in c:\users\shazna salman\appdata\local\programs\python\python311\lib\site-
packages (from opencv-python) (1.26.0)

[notice] A new release of pip is available: 23.3 -> 23.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

- **CVZone**: CVZone is a Python library that enhances OpenCV capabilities, especially for tasks like hand tracking and classification in computer vision projects.

```
C:\Users\shazna salman\AppData\Local\Programs\Python\Python311\Scripts>pip install cvzone
Collecting cvzone
  Downloading cvzone-1.6.1.tar.gz (25 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: opencv-python in c:\users\shazna salman\appdata\local\programs\python\python311\lib\site-
packages (from cvzone) (4.8.1.78)
Requirement already satisfied: numpy in c:\users\shazna salman\appdata\local\programs\python\python311\lib\site-packages
 (from cvzone) (1.26.0)
Building wheels for collected packages: cvzone
  Building wheel for cvzone (setup.py) ... done
  Created wheel for cvzone: filename=cvzone-1.6.1-py3-none-any.whl size=26307 sha256=53bf95995af3848f379c7ca0c7a3316f007
f98981b547479581c3c476cf7cfbf
  Stored in directory: c:\users\shazna salman\appdata\local\pip\cache\wheels\ab\36\ec\47be2d4e59dc4289e684d5b0dde54d1e72
e51a614e57690e85
Successfully built cvzone
Installing collected packages: cvzone
Successfully installed cvzone-1.6.1

[notice] A new release of pip is available: 23.3 -> 23.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

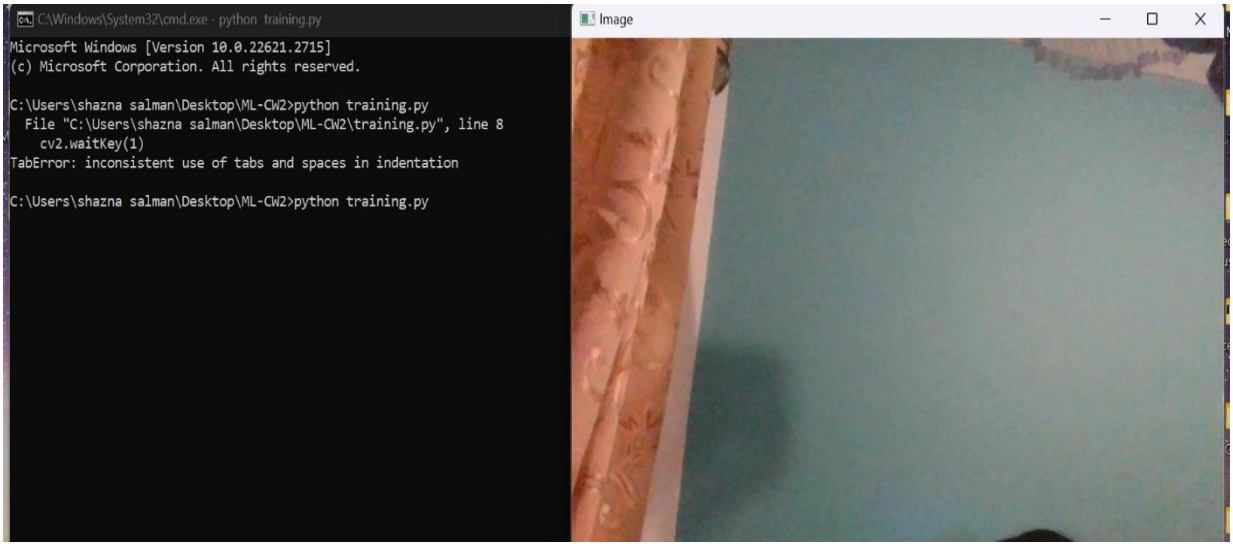
- **Mediapipe**: Developed by Google, Mediapipe is a library for building machine learning applications. It provides pre-trained models for tasks such as hand tracking and simplifies the development of real-time applications.

```
C:\Users\shazna salman\AppData\Local\Programs\Python\Python311\Scripts>pip install mediapipe
Collecting mediapipe
  Downloading mediapipe-0.10.9-cp311-cp311-win_amd64.whl.metadata (9.8 kB)
Requirement already satisfied: absl-py in c:\users\shazna salman\appdata\local\programs\python\python311\lib\site-packag
es (from mediapipe) (2.0.0)
Collecting attrs>=19.1.0 (from mediapipe)
  Downloading attrs-23.1.0-py3-none-any.whl (61 kB)
     ---------------------------------------- 61.2/61.2 kB 808.3 kB/s eta 0:00:00
Requirement already satisfied: flatbuffers>=2.0 in c:\users\shazna salman\appdata\local\programs\python\python311\lib\si
te-packages (from mediapipe) (23.5.26)
Requirement already satisfied: matplotlib in c:\users\shazna salman\appdata\local\programs\python\python311\lib\site-pac
kages (from mediapipe) (3.8.0)
Requirement already satisfied: numpy in c:\users\shazna salman\appdata\local\programs\python\python311\lib\site-packages
 (from mediapipe) (1.26.0)
Collecting opencv-contrib-python (from mediapipe)
  Downloading opencv_contrib_python-4.8.1.78-cp37-abi3-win_amd64.whl.metadata (20 kB)
Collecting protobuf<4,>=3.11 (from mediapipe)
  Downloading protobuf-3.20.3-py2.py3-none-any.whl (162 kB)
     ---------------------------------------- 162.1/162.1 kB 1.6 MB/s eta 0:00:00
Collecting sounddevice>=0.4.4 (from mediapipe)
  Downloading sounddevice-0.4.6-py3-none-win_amd64.whl (199 kB)
     ---------------------------------------- 199.7/199.7 kB 2.0 MB/s eta 0:00:00
Collecting CFFI>=1.0 (from sounddevice>=0.4.4->mediapipe)
  Downloading cffi-1.16.0-cp311-cp311-win_amd64.whl.metadata (1.5 kB)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\shazna salman\appdata\local\programs\python\python311\lib\si
te-packages (from matplotlib->mediapipe) (1.1.1)
Requirement already satisfied: cycler>=0.10 in c:\users\shazna salman\appdata\local\programs\python\python311\lib\site-p
ackages (from matplotlib->mediapipe) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\shazna salman\appdata\local\programs\python\python311\lib\s
```

2.2. Camera Check

The code snippet continuously displays the live video feed from the camera to check its functionality. If the camera is accessible, a window shows the real-time video stream. This practice ensures the camera setup is correct before moving to more complex tasks like hand tracking or image processing.

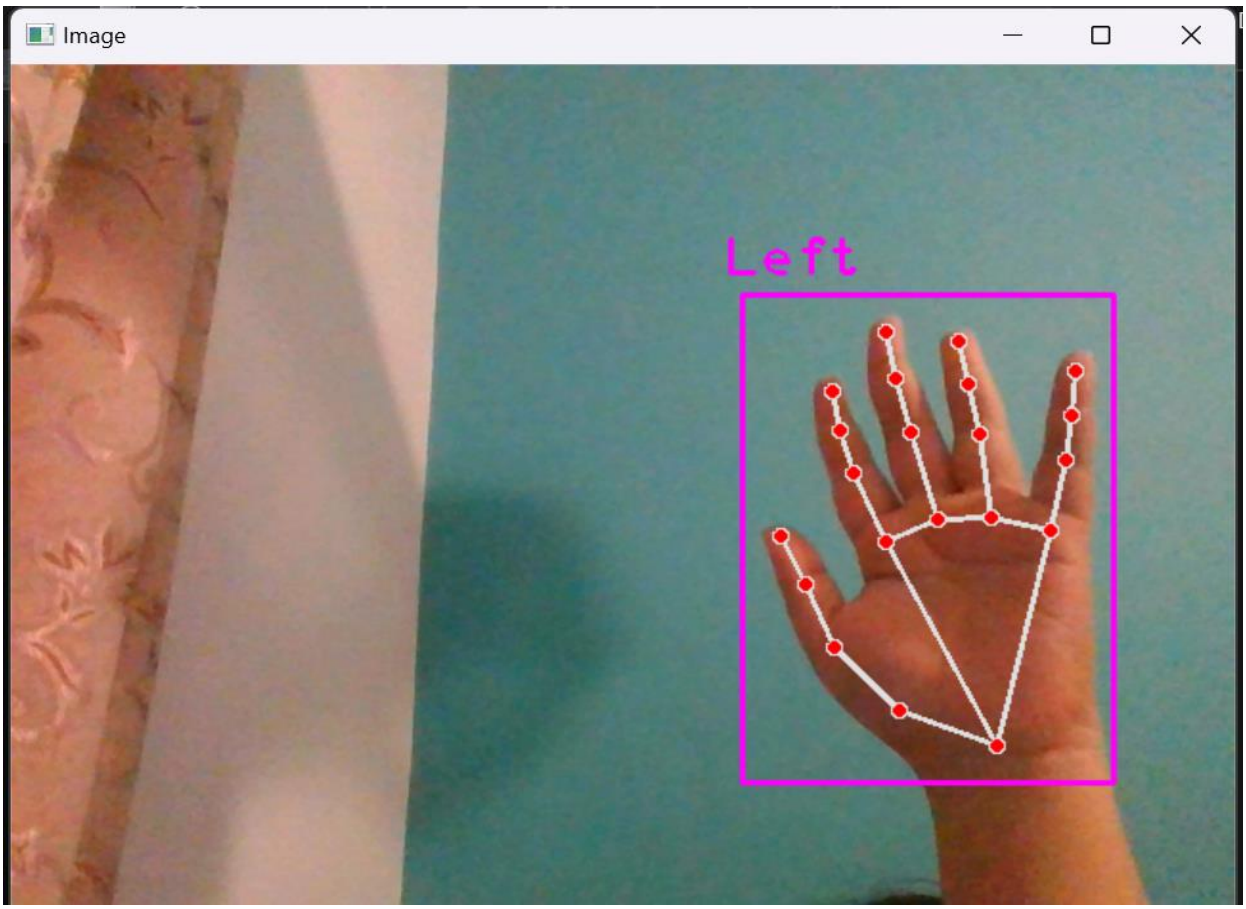
```
import cv2

cap = cv2.VideoCapture(0)

while True:
    success, img = cap.read()
    cv2.imshow("Image", img)
    cv2.waitKey(1)
```



2.3. Hand Tracking

The hand detection and tracking code utilizes the "HandTrackingModule" to continuously analyze video frames from the webcam, identifying and tracking hands based on landmarks and bounding boxes. This real-time tracking capability forms a crucial component for subsequent gesture recognition and sign language interpretation in the project.

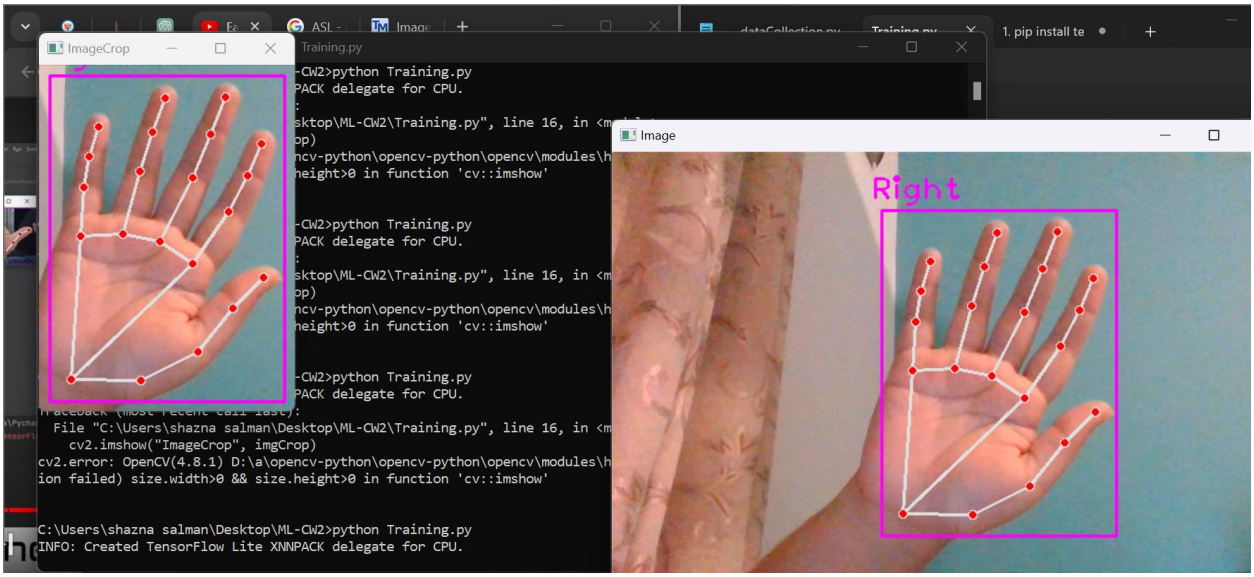
```
import cv2
from cvzone.HandTrackingModule import HandDetector


cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
while True:
    success, img = cap.read()
    hands, img = detector.findHands(img)
    cv2.imshow("Image", img)
    cv2.waitKey(1)
```



2.4. Image Cropping

This code snippet focuses on isolating the region of interest by cropping the captured image based on the detected hand's bounding box. By doing so, it extracts the hand gesture from the overall frame, allowing for a more precise analysis of the hand's configuration and movements. This cropped image becomes a crucial input for further processing, such as resizing and adapting the image to fit the dimensions required for accurate gesture classification.

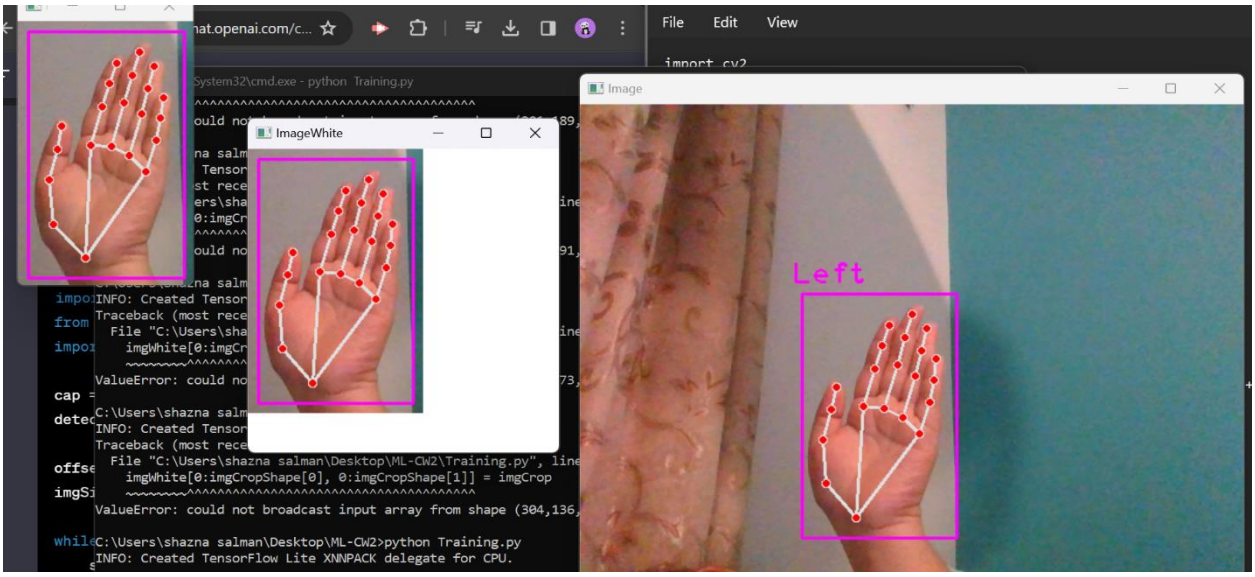
```
import cv2
from cvzone.HandTrackingModule import HandDetector

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

offset = 30

while True:
    success, img = cap.read()
    hands, img = detector.findHands(img)
    if hands:
        hand = hands[0]
        x, y, w, h = hand['bbox']
        imgCrop = img[y - offset : y + h + offset, x - offset : x + w +
offset]
        cv2.imshow("ImageCrop", imgCrop)

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



2.5. Creating White Image Matrix

This section of the code is dedicated to creating a white image matrix, essentially a blank canvas, with the same predefined dimensions. The cropped hand gesture is then superimposed onto this white matrix. The purpose of this step is to standardize the input size for the gesture recognition model. By placing the cropped image onto the white matrix, the system ensures consistency in the dimensions of the input data, facilitating effective training and prediction processes. This matrix manipulation helps prepare the data for accurate interpretation by the machine learning model.

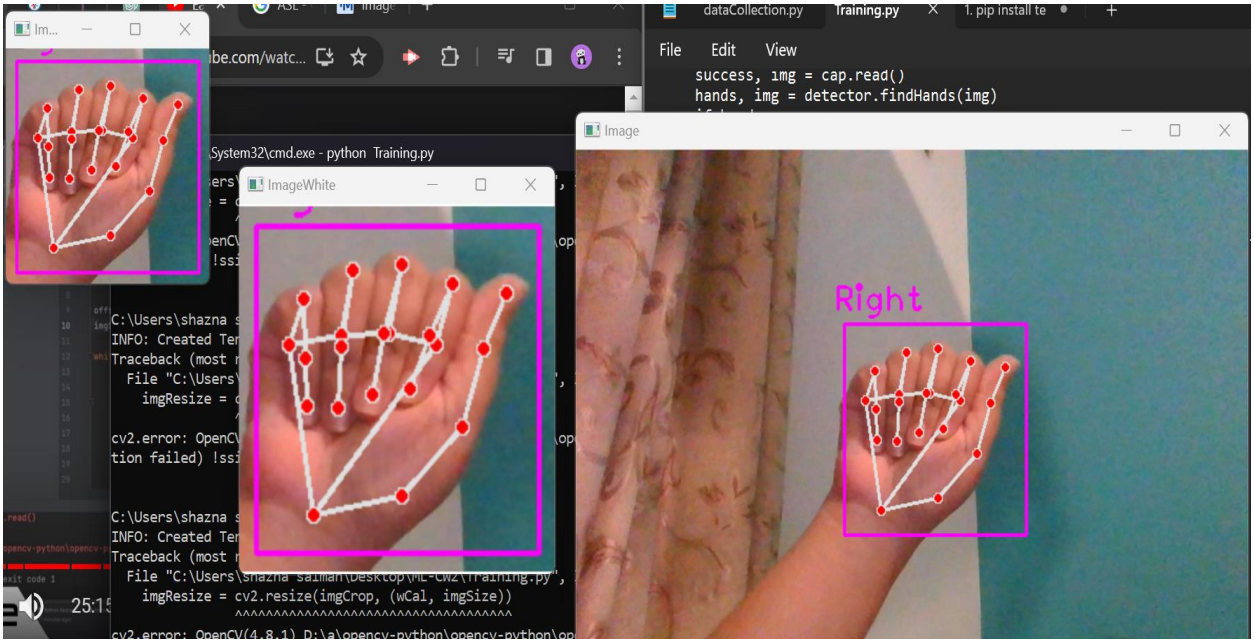
```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np

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

offset = 30
imgSize = 300

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
        imgWhite[0:imgCropShape[0], 0:imgCropShape[1]] = imgCrop

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

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

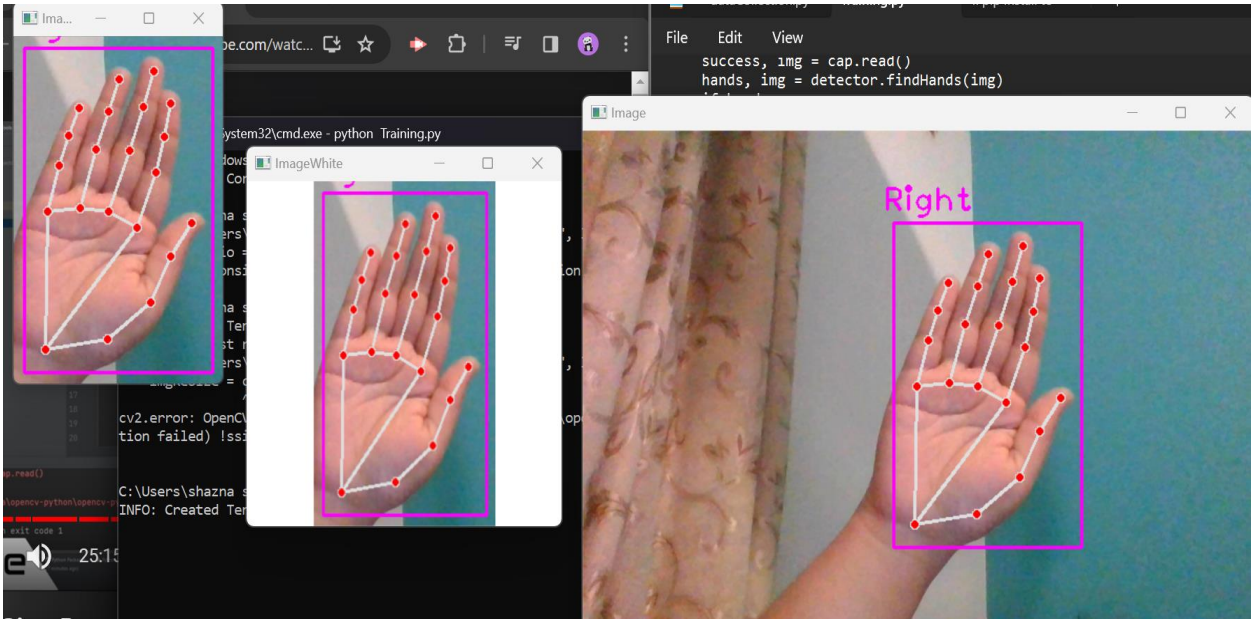


2.6. Proper Image Fitting

This section ensures the seamless integration of the cropped hand gesture into a designated white canvas. It dynamically adjusts the size and positioning based on the aspect ratio of the hand bounding box. Whether the gesture is vertically or horizontally oriented, the code smartly resizes and centers the image within the white canvas. This step is crucial for maintaining the gesture's proportions while neatly fitting it into the allocated space.

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

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

offset = 30
imgSize = 300

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)
    cv2.waitKey(1)
```





It is clear that the cropped image is fitted properly according to the size and motions made.

The change is elaborated as the images given above.

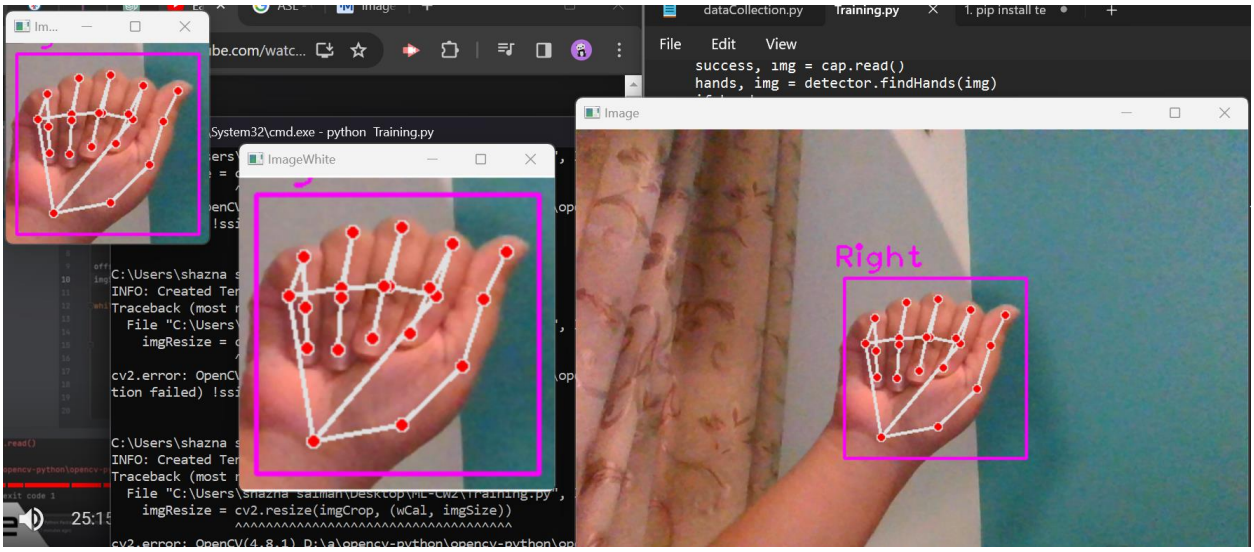
2.7. Dataset Creation

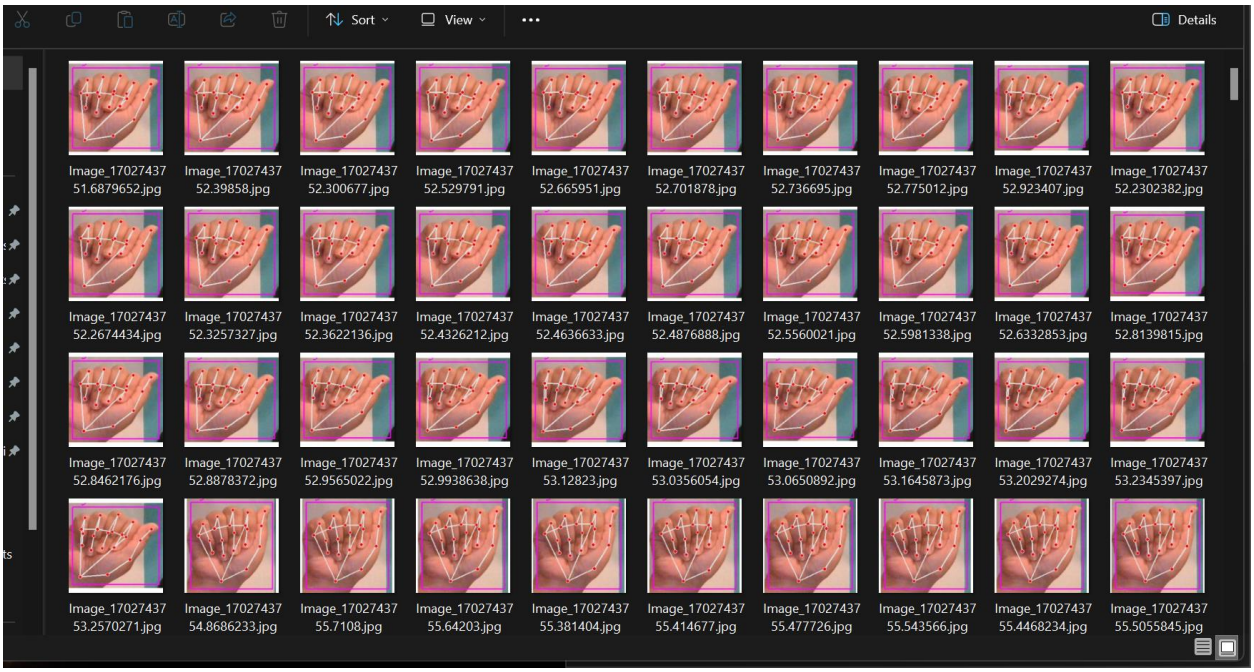
This segment of the project involves capturing and compiling a dataset for training the model. The code (training.py) operates the camera to continuously capture images of hand gestures when prompted by the user pressing the "a" key. These images are then stored in the specified dataset folder for each corresponding gesture category. The dataset creation is an essential step as it provides diverse examples for the model to learn from, enhancing its ability to accurately recognize and classify different sign language gestures.

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

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

offset = 30
imgSize = 300

folder = r"C:\Users\shazna salman\Desktop\ML-CW2\dataset\A"
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("a"):
        counter += 1
        cv2.imwrite(f'{folder}/Image_{time.time()}.jpg', imgWhite)
        print(counter)
```

(a) This is **A** in sign language. When the “a” button is clicked, the image is saved in the path given.

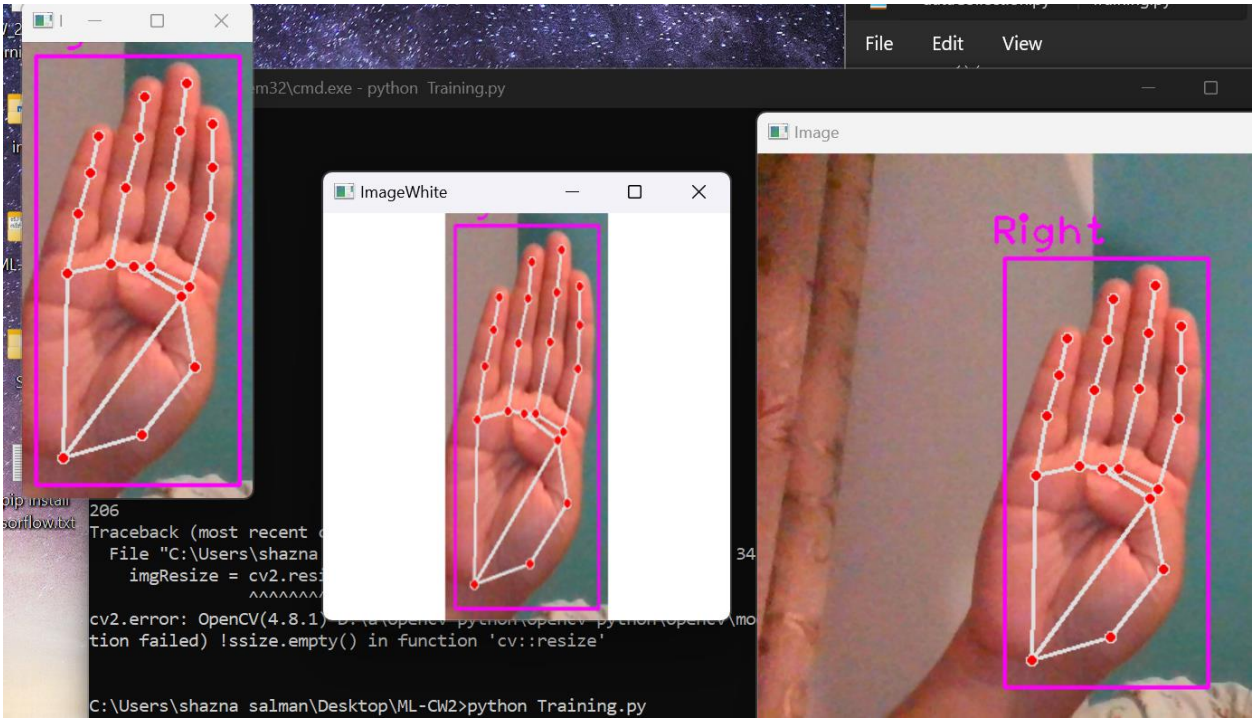
C:\Users\shazna salman\Desktop\ML-CW2\dataset\A"

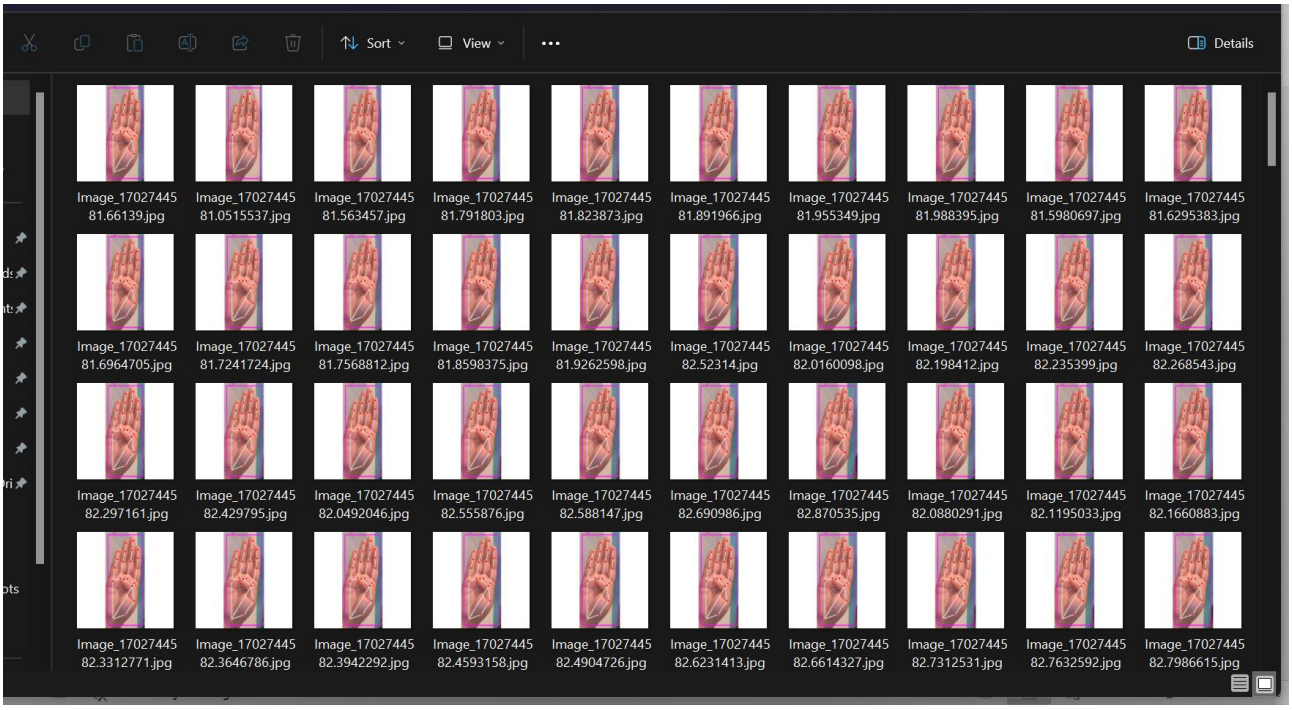




(b) This is **B** in sign language. When the “a” button is clicked, the image is saved in the path given.

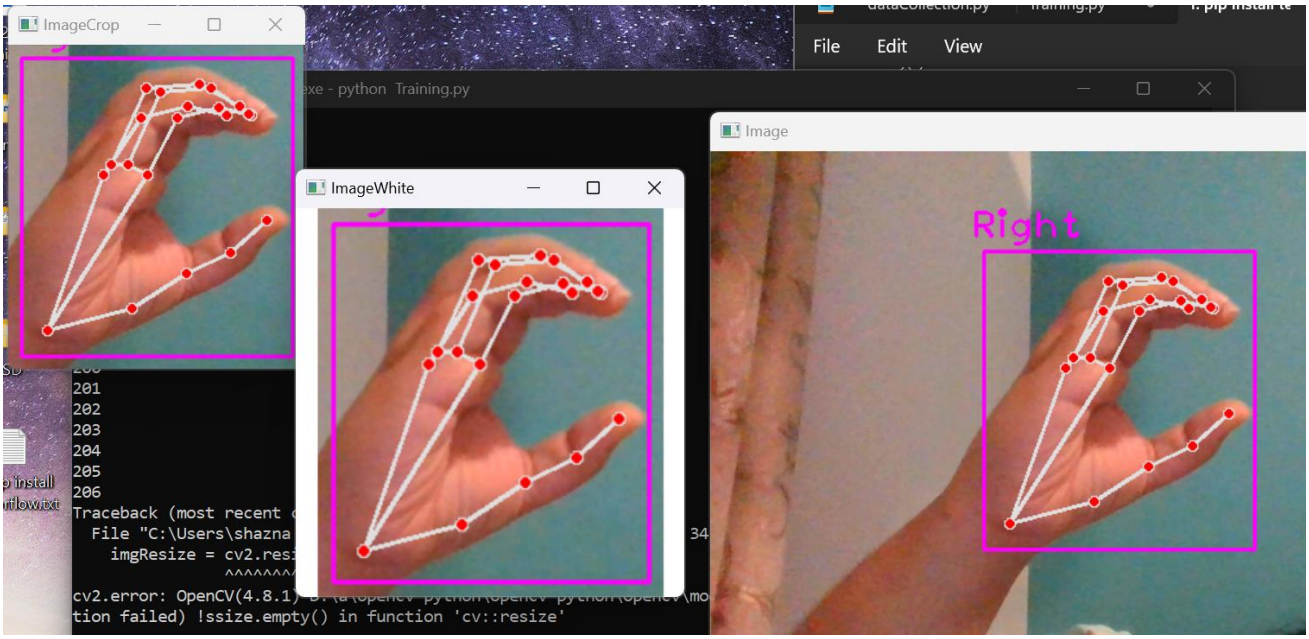
C:\Users\shazna salman\Desktop\ML-CW2\dataset\B"

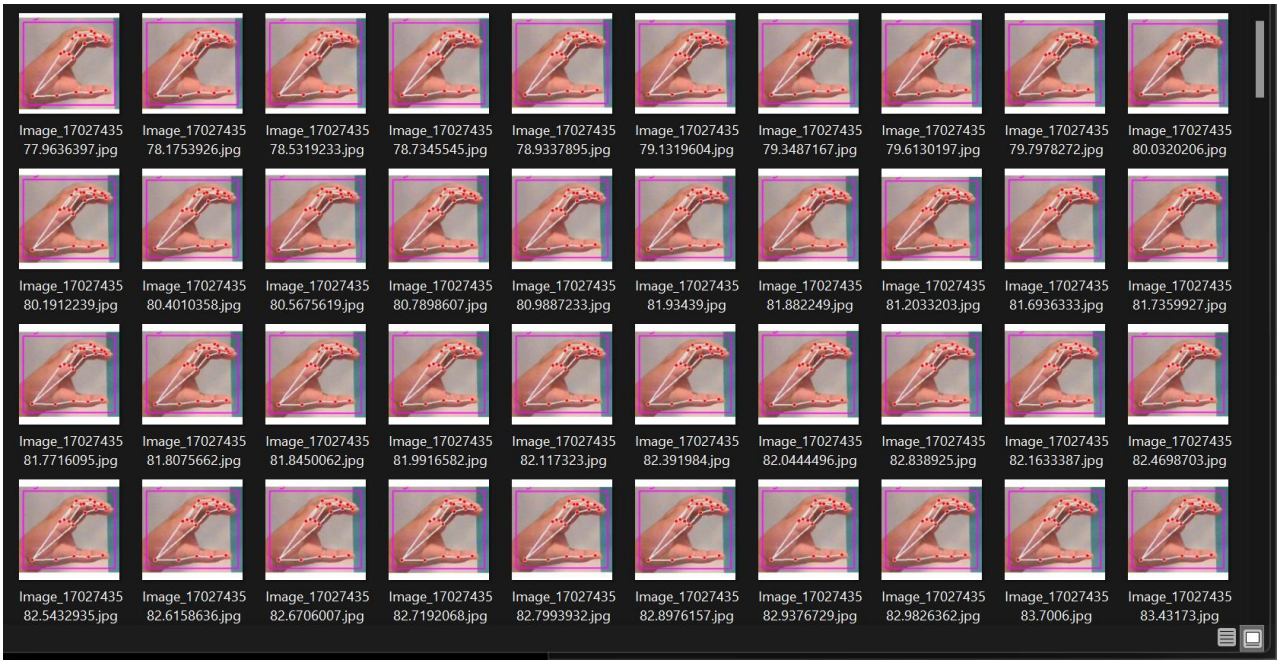




(c) This is **C** in sign language. When the "a" button is clicked, the image is saved in the path given.

C:\Users\shazna salman\Desktop\ML-CW2\dataset\C"



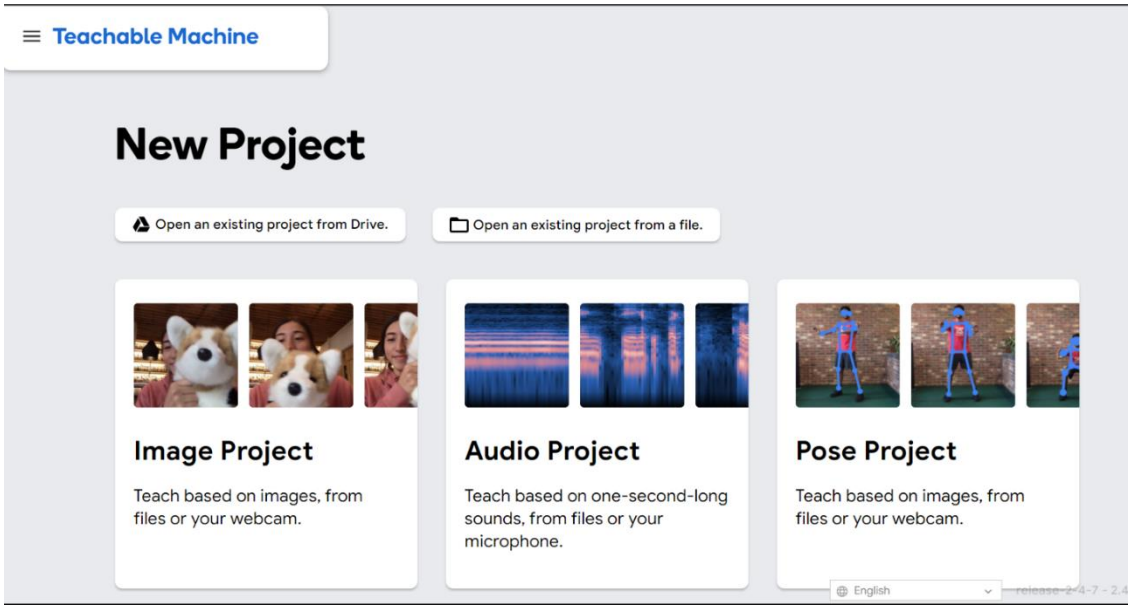


Similarly, all 26 letters of the alphabet are organized.

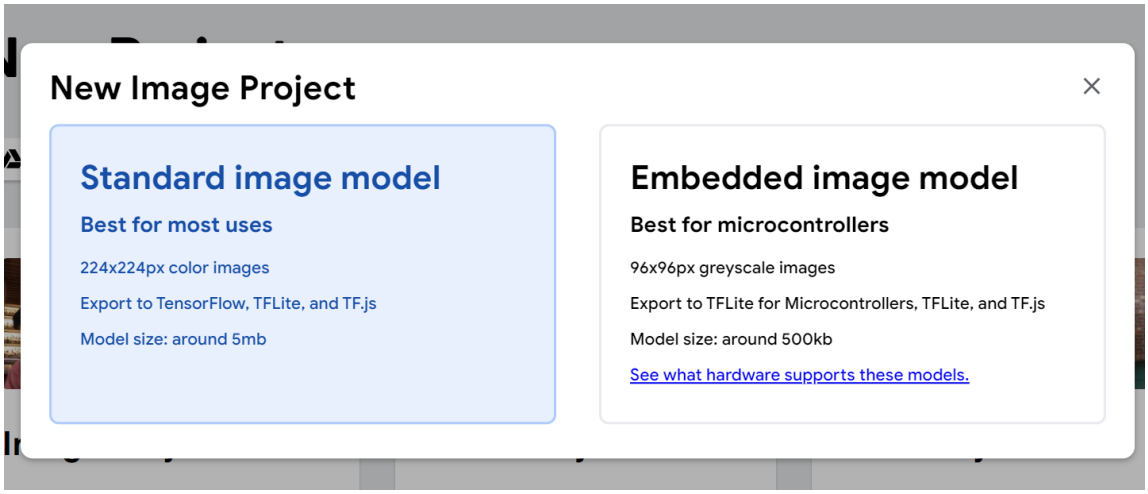
2.8. Train the data using teachable machine

Teachable Machine simplifies the process of training a machine learning model by breaking it down into user-friendly steps. First, users upload their dataset, consisting of labeled images representing different sign language gestures. Next, configure the training parameters, such as the number of classes and training duration. The platform then initiates the training process, leveraging TensorFlow under the hood. Once training is complete, users can preview and evaluate the model's performance. Finally, the trained model files, including the architecture and weights, are available for download. This user-friendly approach enables individuals, even with minimal machine learning experience, to create and deploy models for sign language recognition.

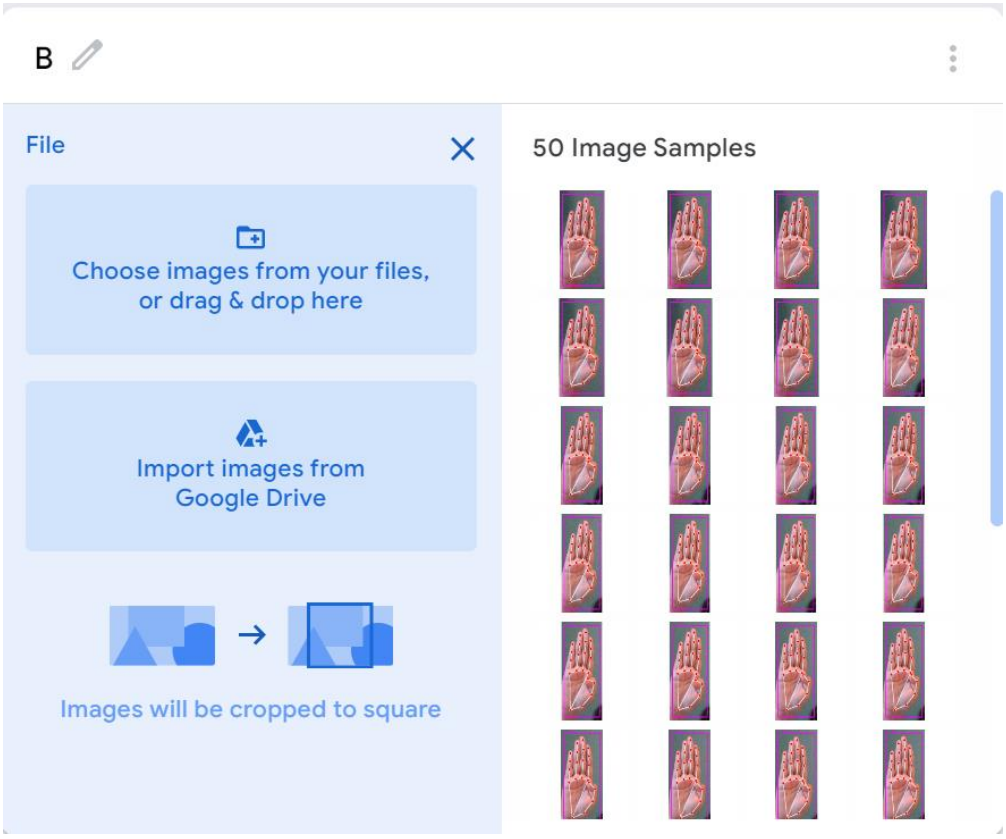
1. Choose a new Image project.



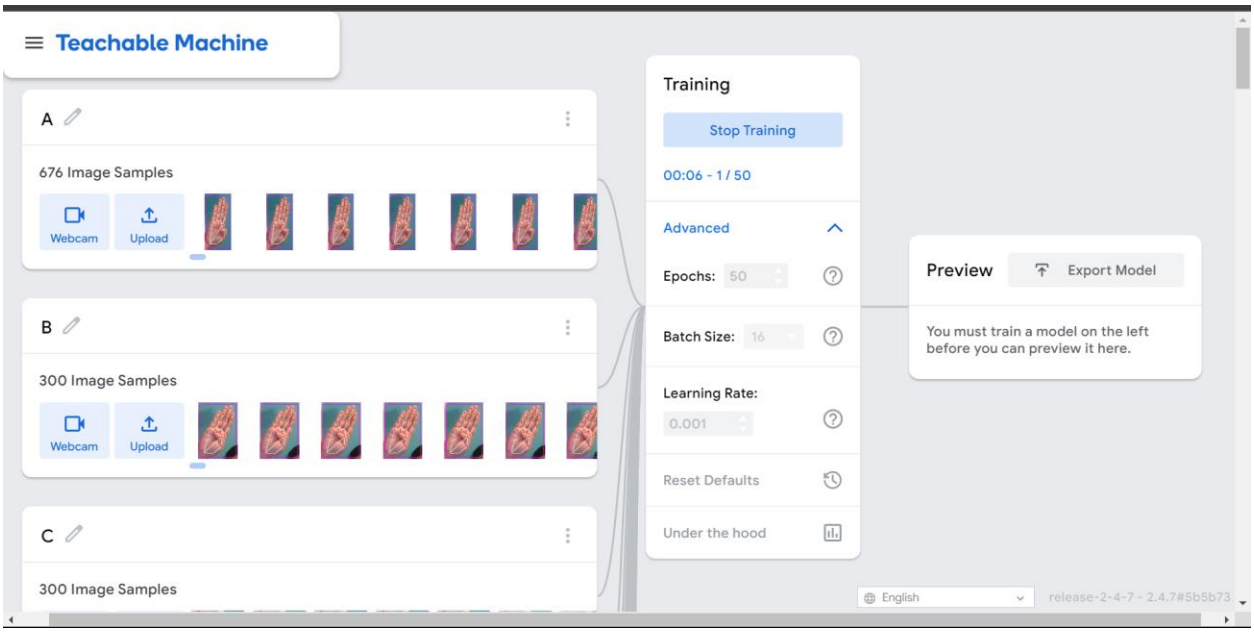
2. Choose Standard image model



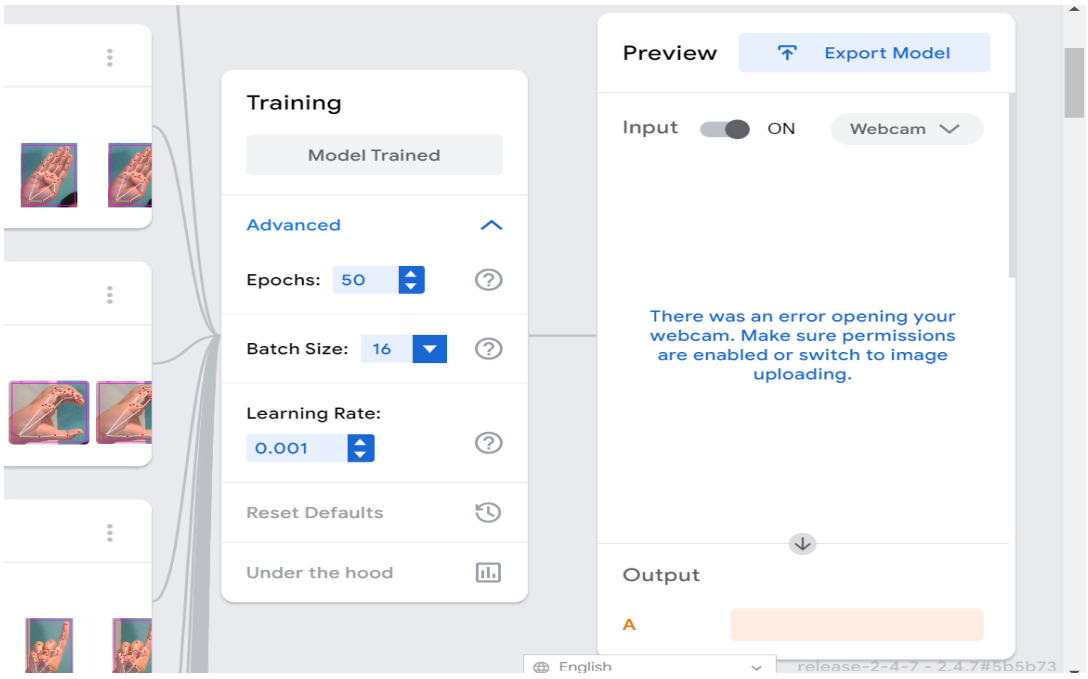
3. **Label the set and choose images and drop them.**



4. **Train the model after adding all the data and labels.**



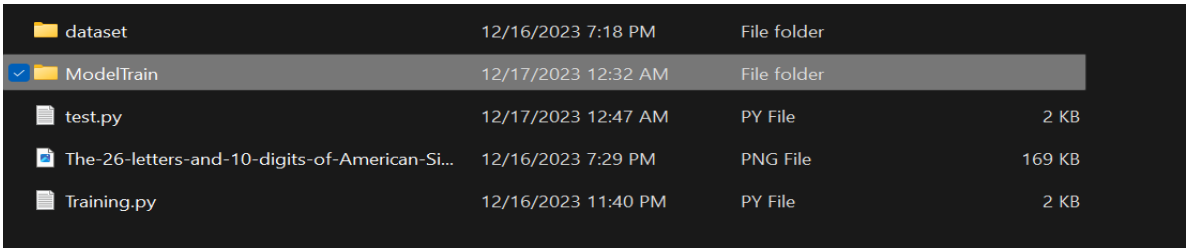
5. **After the training is complete, export the model.**



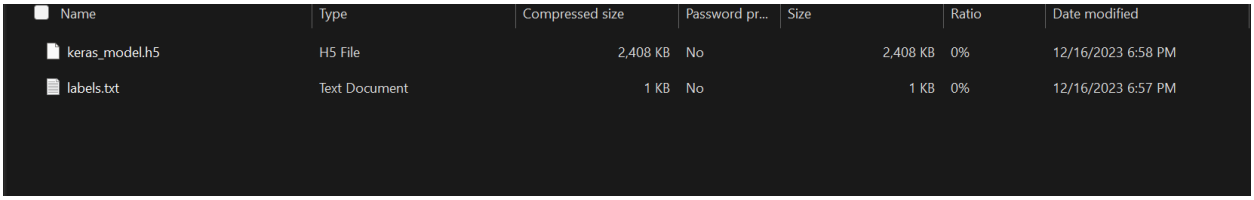
6. **Export the Tensorflow version of the model.**

2.9. Export the model

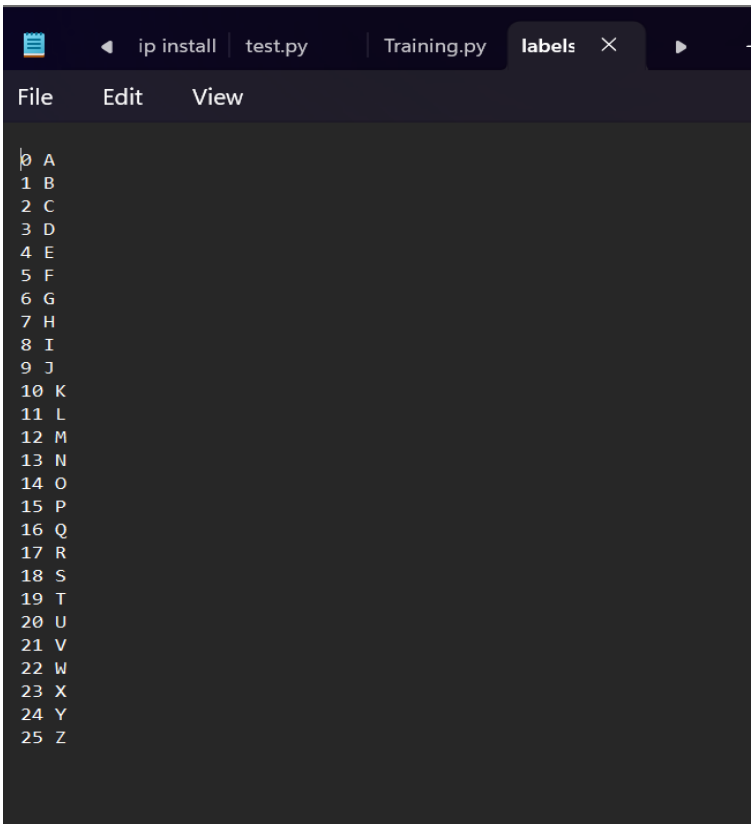
1. **This is the folder where the files related to the model is exported**



2. **These are the files;**



3. **These are the labels which belong to the respective alphabetical letter.**



II. Testing the program (test.py)

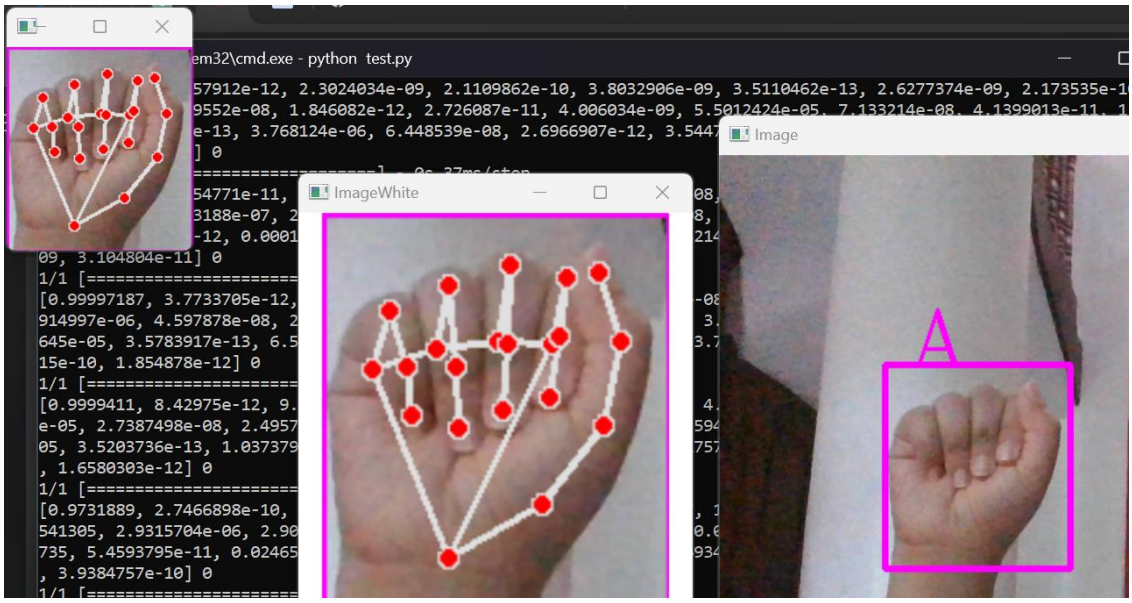
In this testing phase, the program's ability to recognize and display sign language letters is evaluated. As the system captures live video feed, it employs hand tracking to detect and isolate hand gestures. The cropped image of the gesture is then processed and fed into the trained classifier. The program displays the original cropped gesture alongside a white matrix representation. Simultaneously, it predicts the corresponding sign language letter using the trained model. The output includes the recognized letter superimposed on the video feed, providing a real-time demonstration of the system's accuracy in translating sign language gestures into written text. This phase is crucial for assessing the practicality and effectiveness of the developed sign language interpretation system.

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

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

offset = 30
imgSize = 300

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)
    cv2.waitKey(1)
```

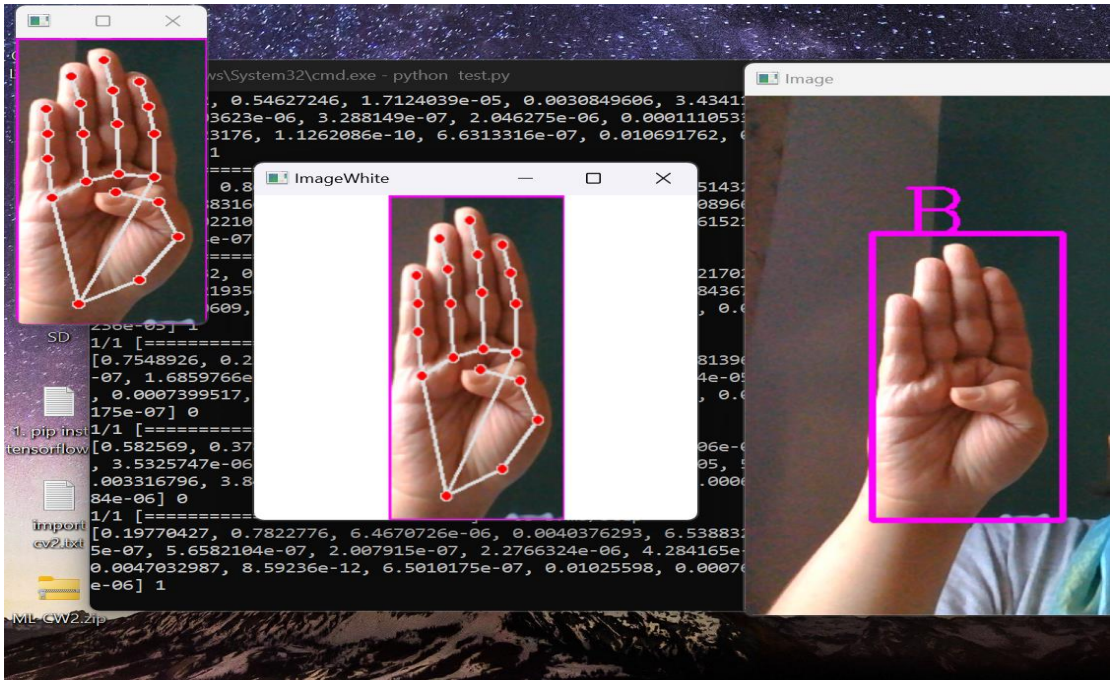
3. OUTPUTS

The various outputs for all 26 letters of the alphabet are given below. The gestures are made in various backgrounds.

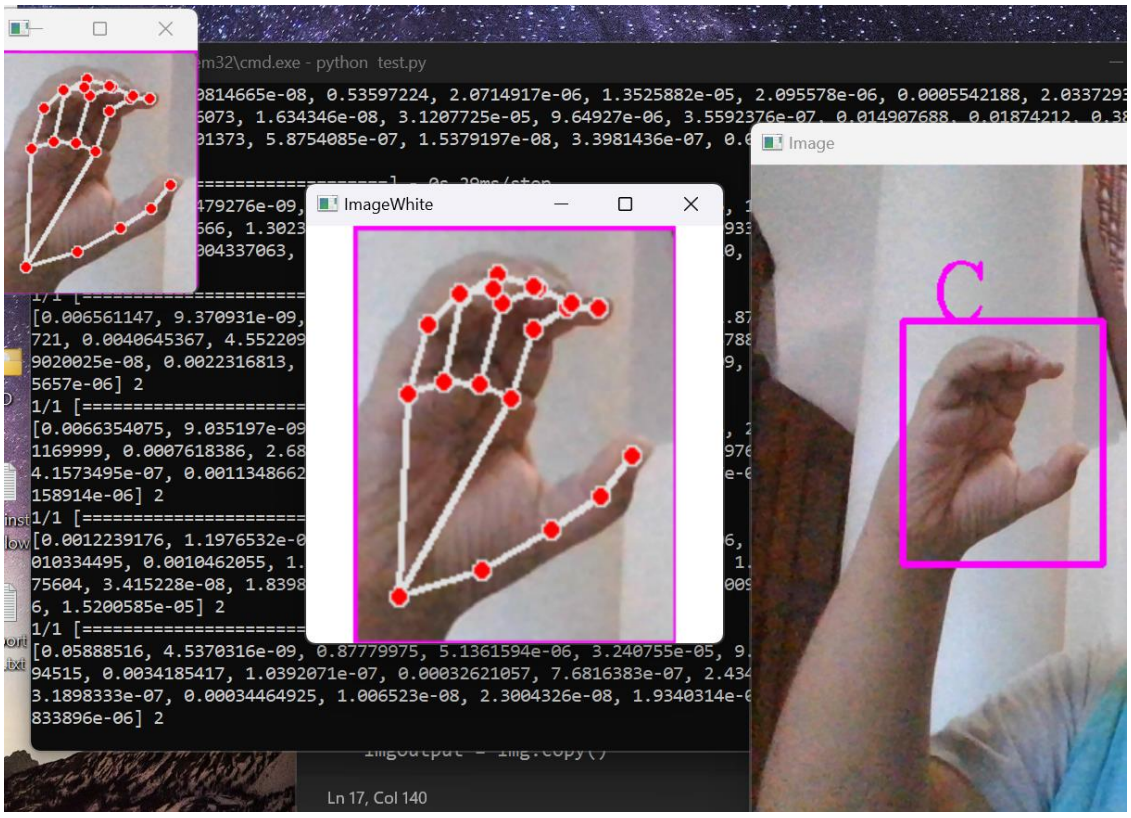
1. A



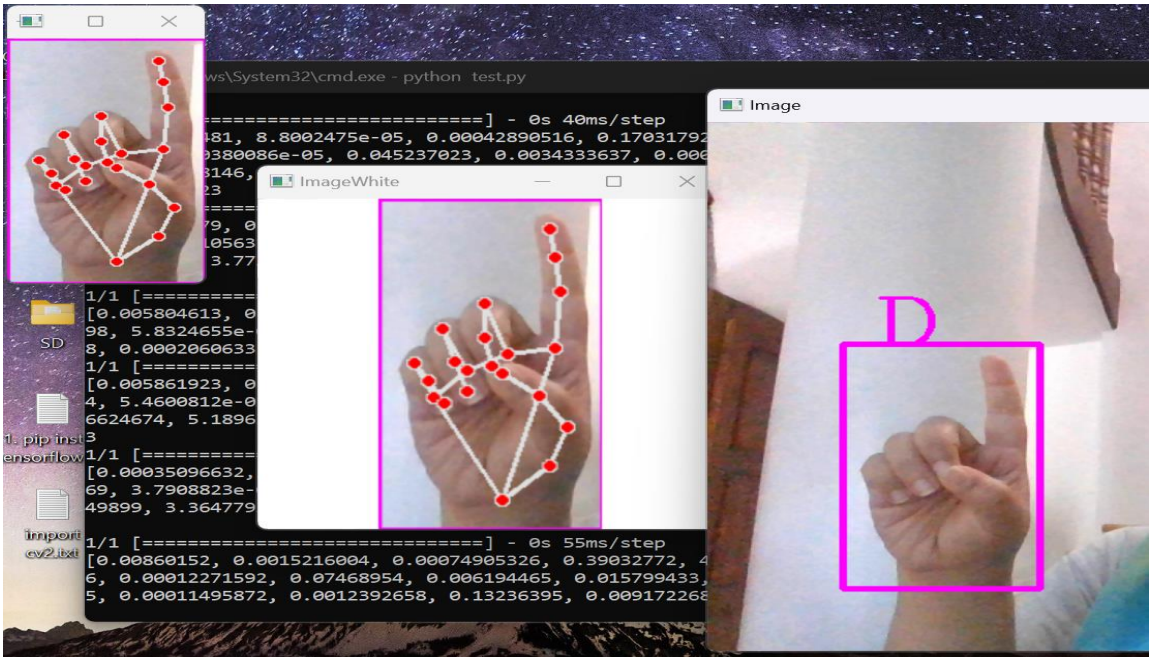
2. B



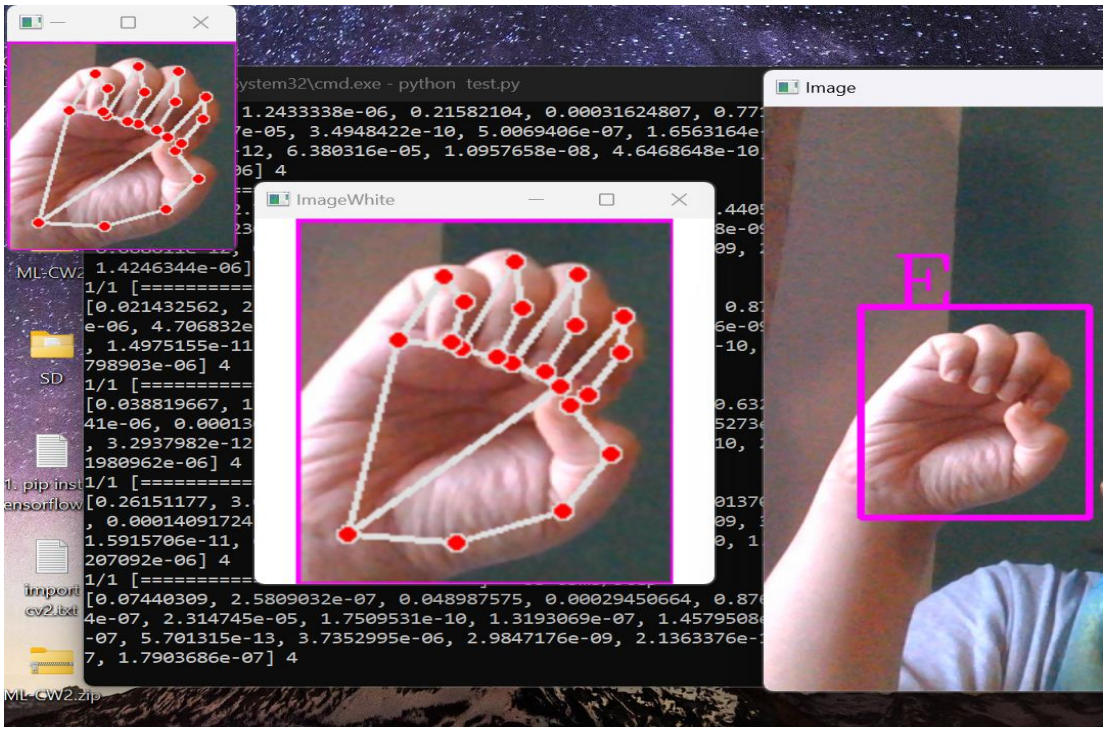
3. C



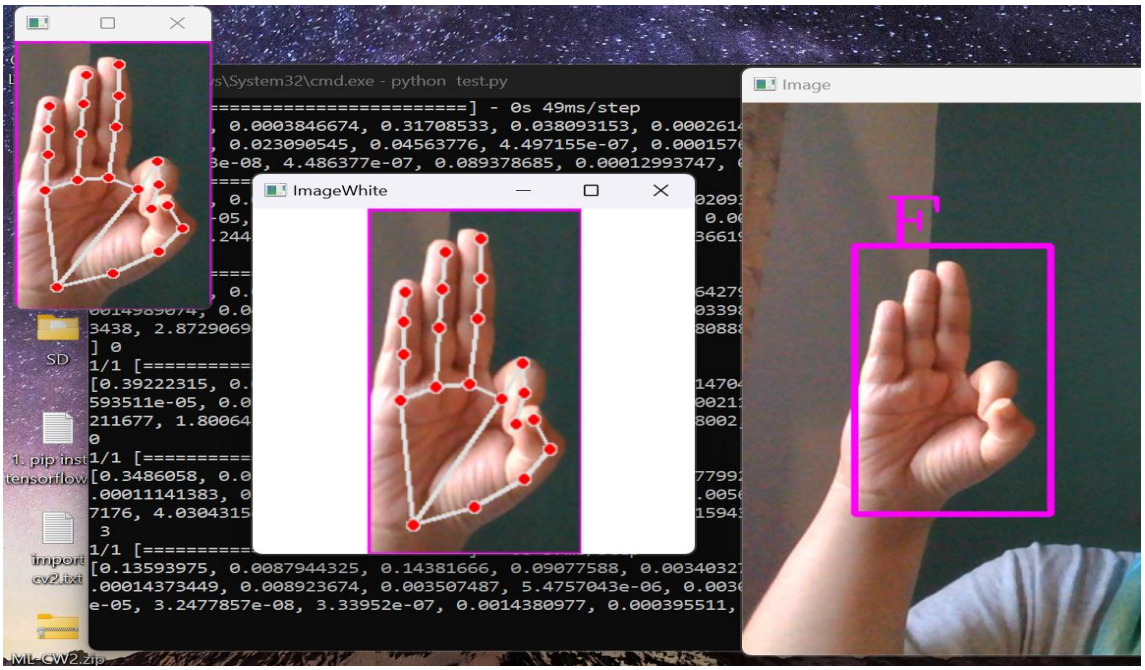
4. D



5. E


ystem32\cmd.exe - python test.py
ImageWhite
Image
F

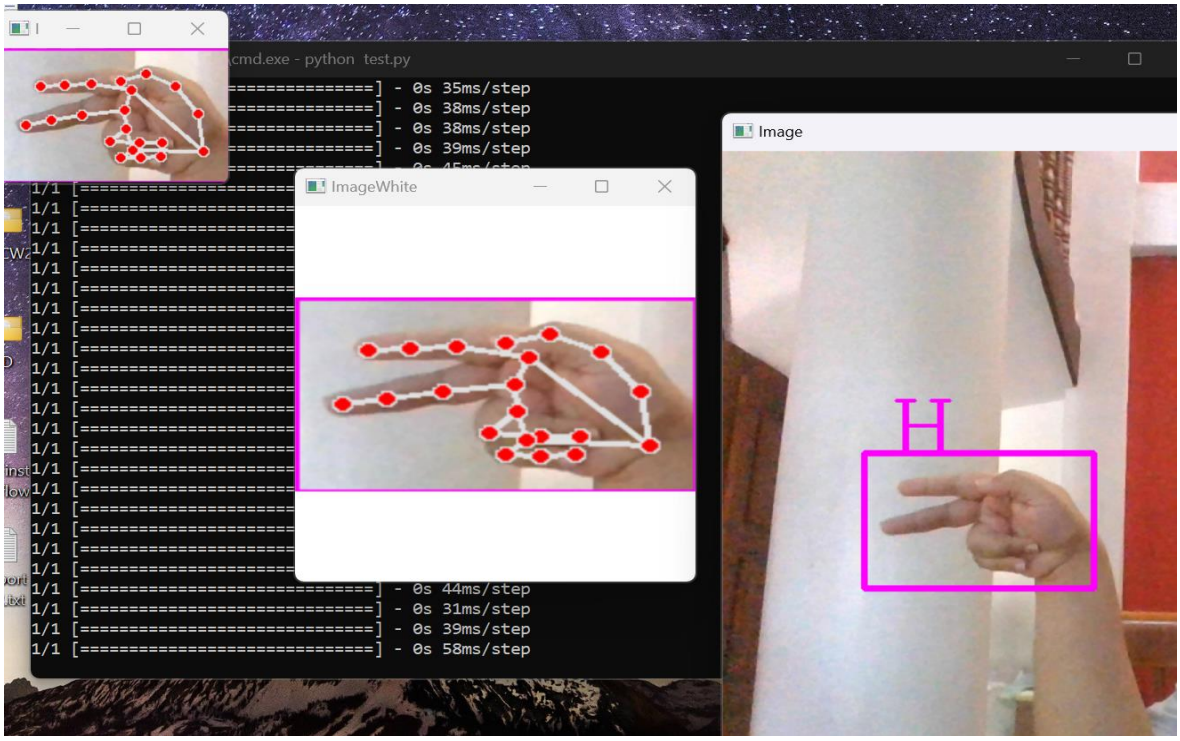
6. F


s\System32\cmd.exe - python test.py
ImageWhite
Image
F

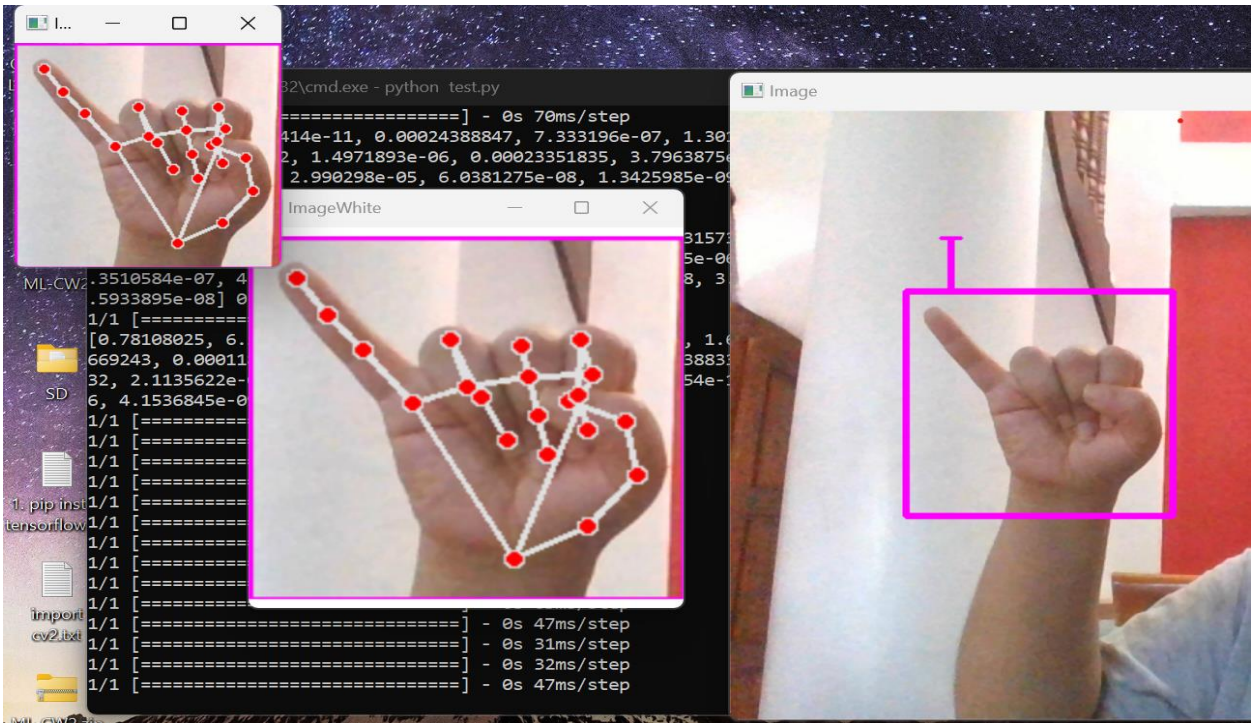
7. G


em32\cmd.exe - python test.py
] - 0s 48ms/step
] - 0s 55ms/step
] - 0s 55ms/step
] - 0s 45ms/step
ImageWhite
Image
G
] - 0s 47ms/step
] - 0s 63ms/step
] - 0s 47ms/step
] - 0s 47ms/step

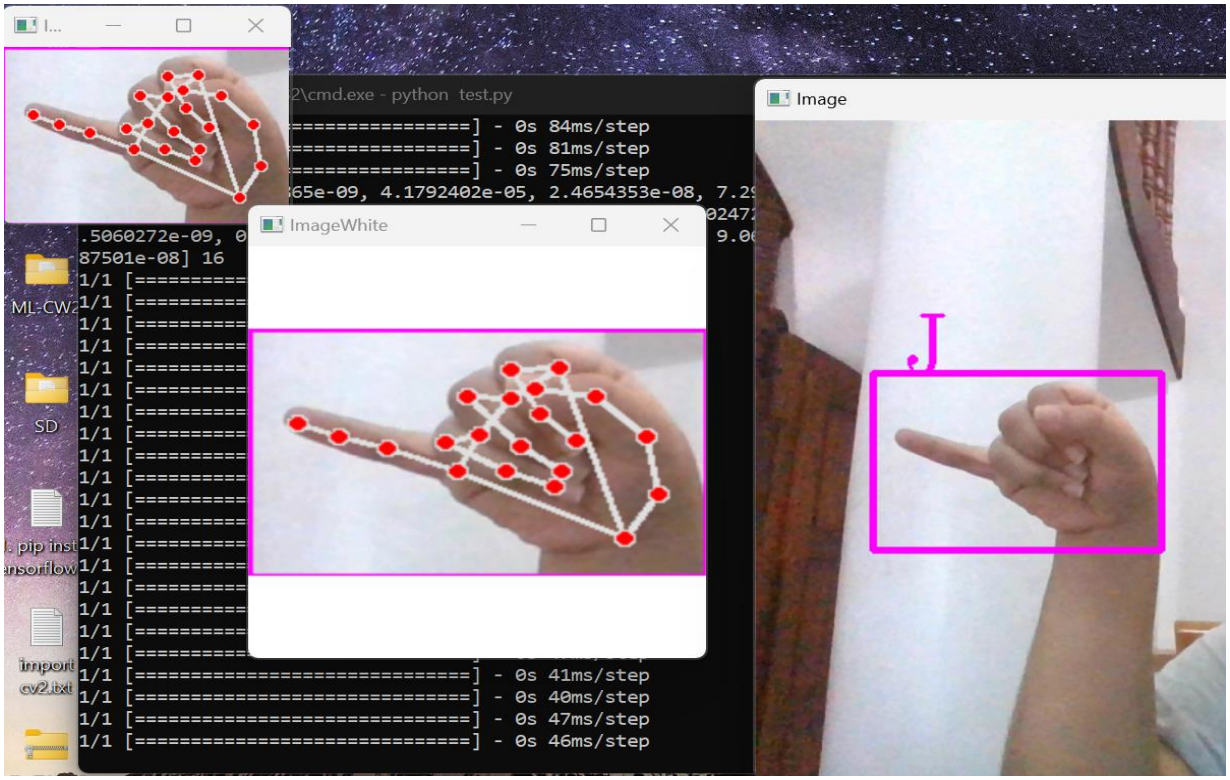
8. H


cmd.exe - python test.py
] - 0s 35ms/step
] - 0s 38ms/step
] - 0s 38ms/step
] - 0s 39ms/step
ImageWhite
Image
H
] - 0s 44ms/step
] - 0s 31ms/step
] - 0s 39ms/step
] - 0s 58ms/step

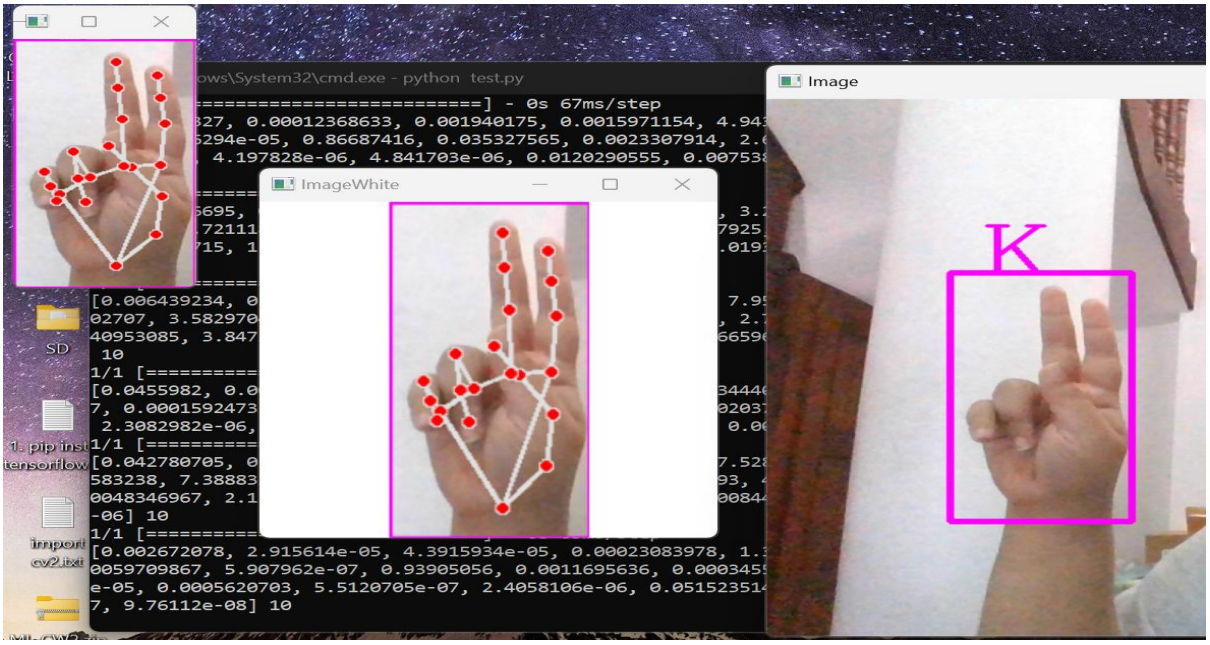
9. I


ImageWhite
Image
1/1 [==========] - 0s 70ms/step
1/1 [==========] - 0s 47ms/step
1/1 [==========] - 0s 31ms/step
1/1 [==========] - 0s 32ms/step
1/1 [==========] - 0s 47ms/step

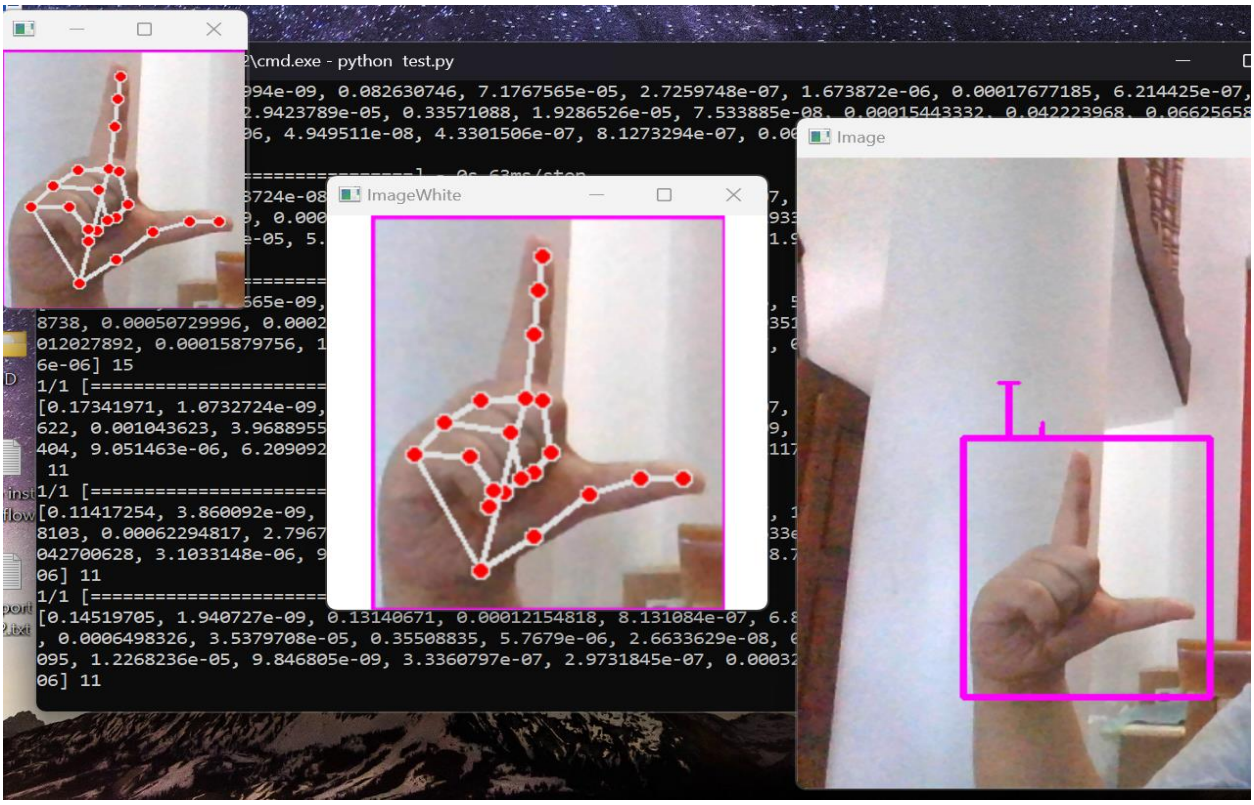
10.J


ImageWhite
Image
1/1 [==========] - 0s 84ms/step
1/1 [==========] - 0s 81ms/step
1/1 [==========] - 0s 75ms/step
1/1 [==========] - 0s 41ms/step
1/1 [==========] - 0s 40ms/step
1/1 [==========] - 0s 47ms/step
1/1 [==========] - 0s 46ms/step

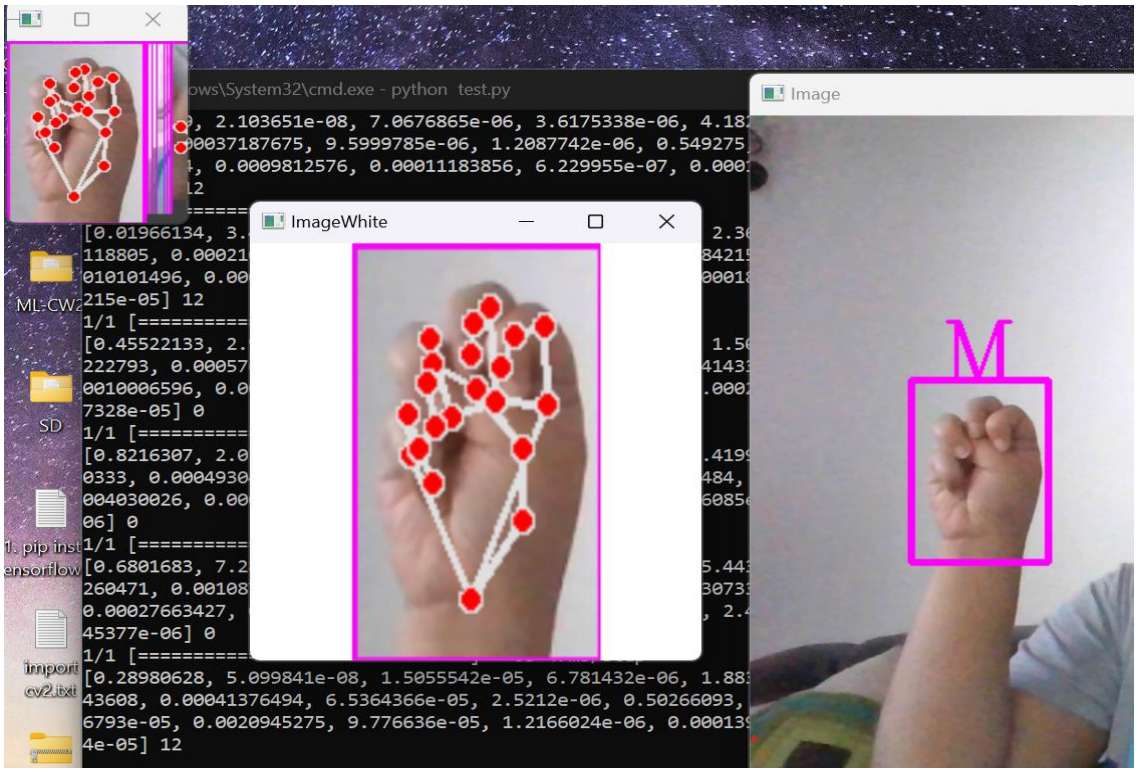
11.K



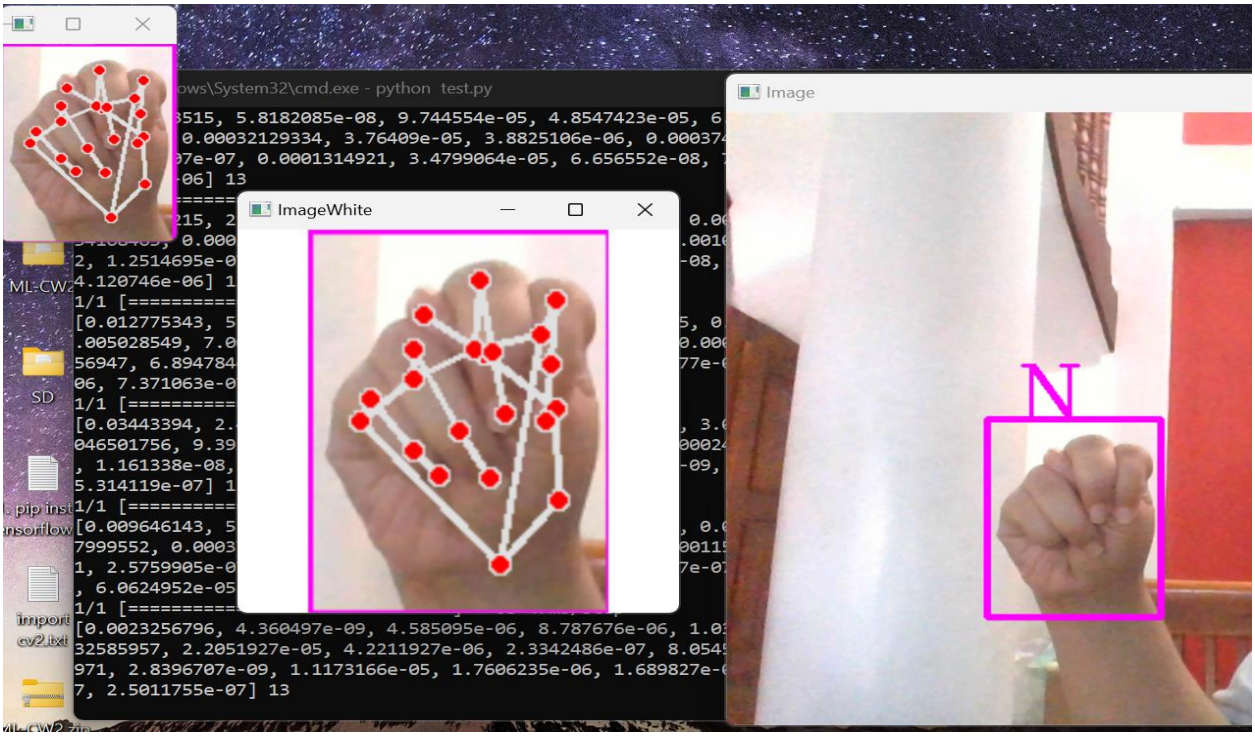
12.L



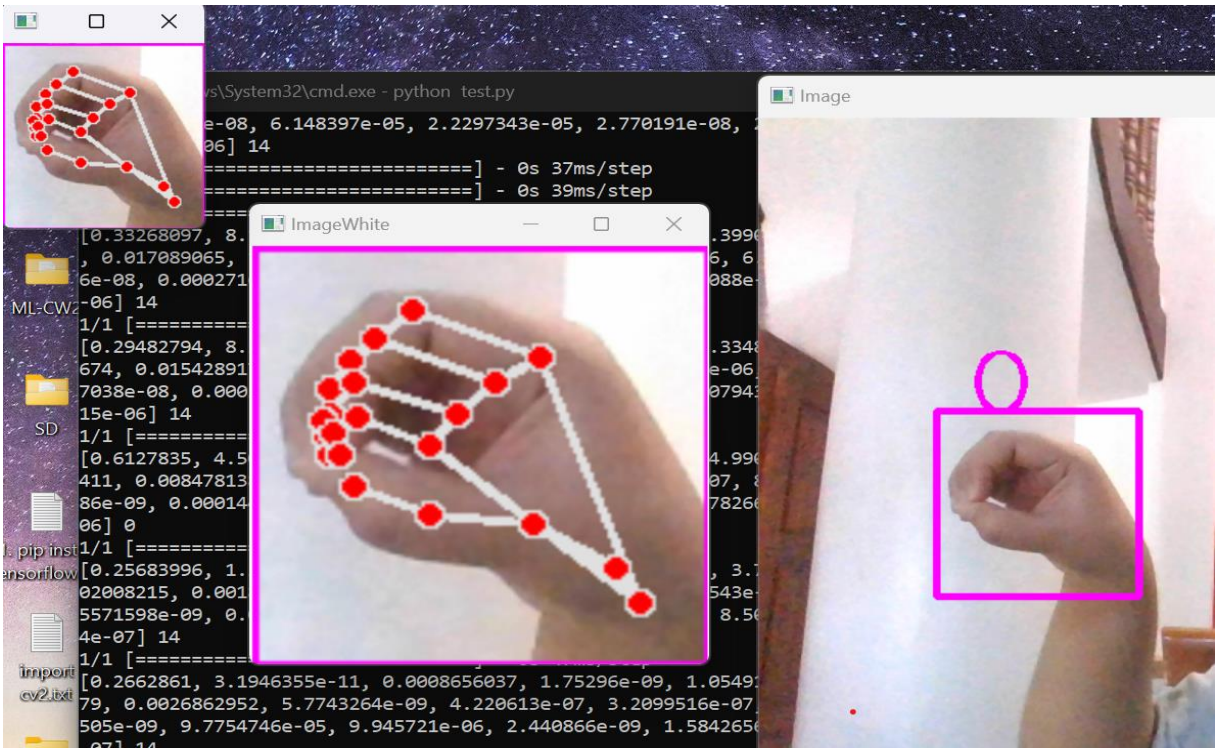
13.M


ows\System32\cmd.exe - python test.py
ImageWhite
Image
M

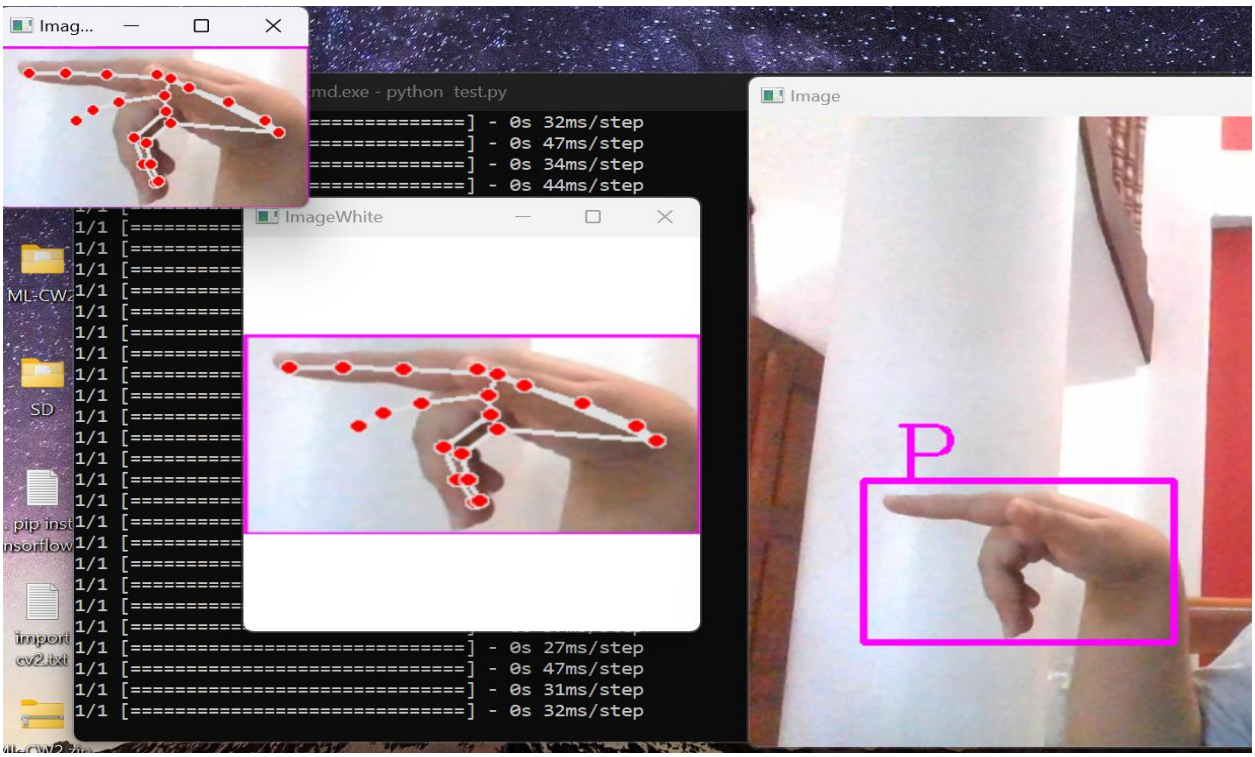
14.N


ows\System32\cmd.exe - python test.py
ImageWhite
Image
N

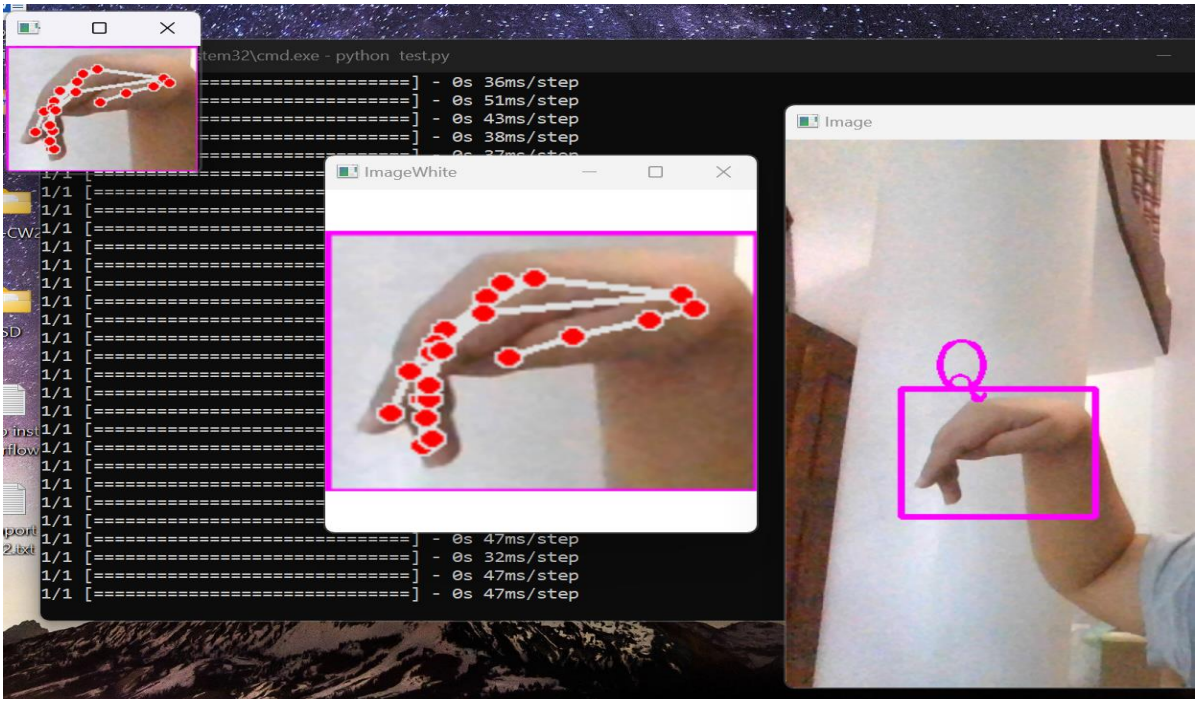
15.O



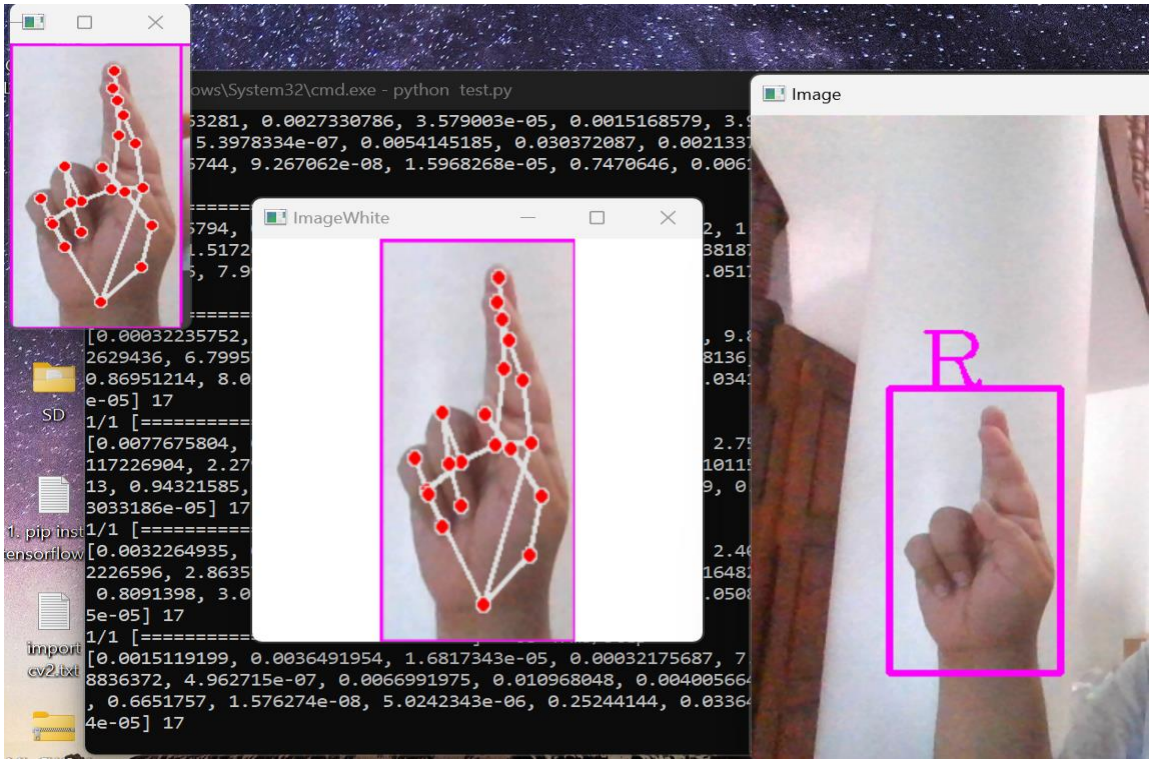
16.P



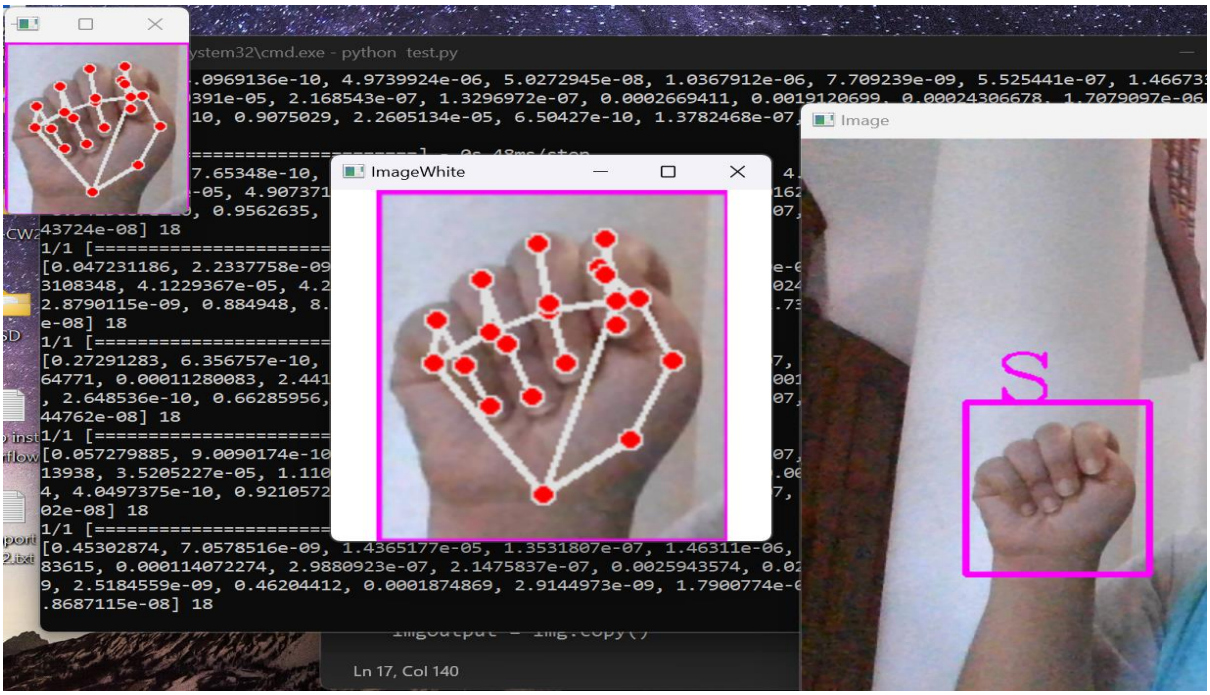
17.Q



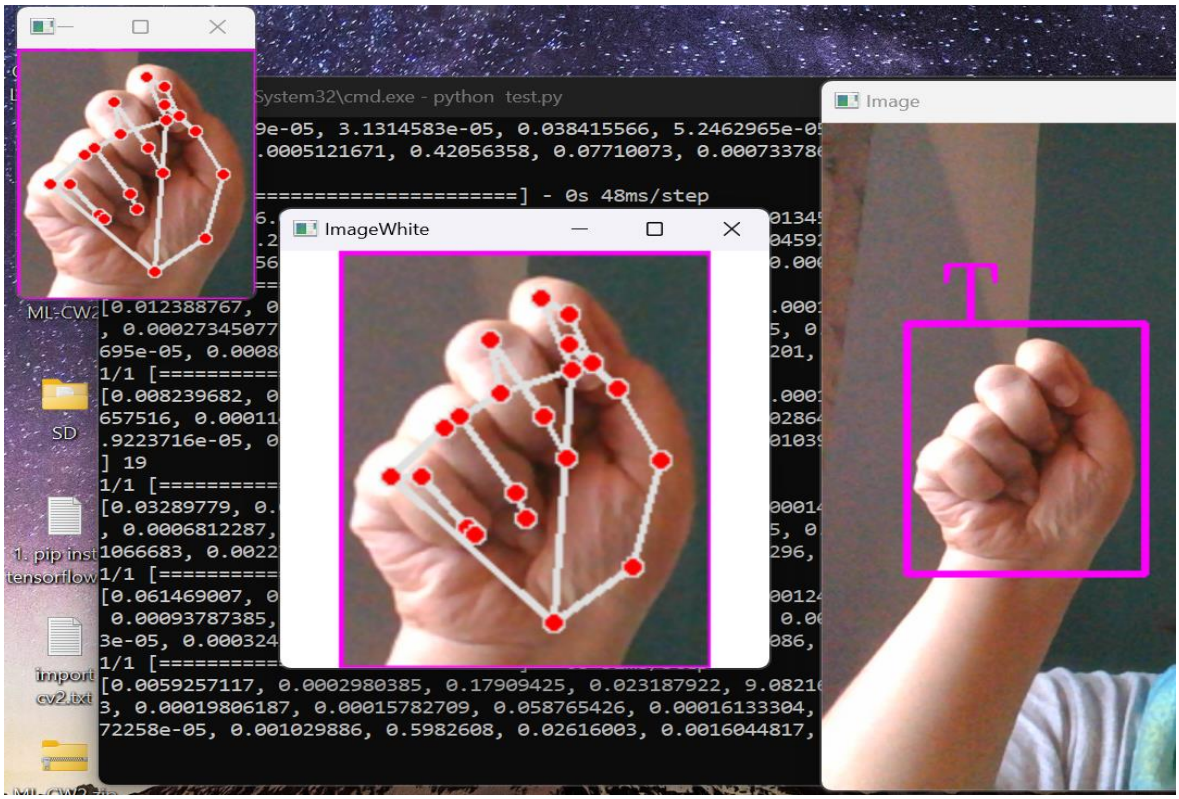
18.R



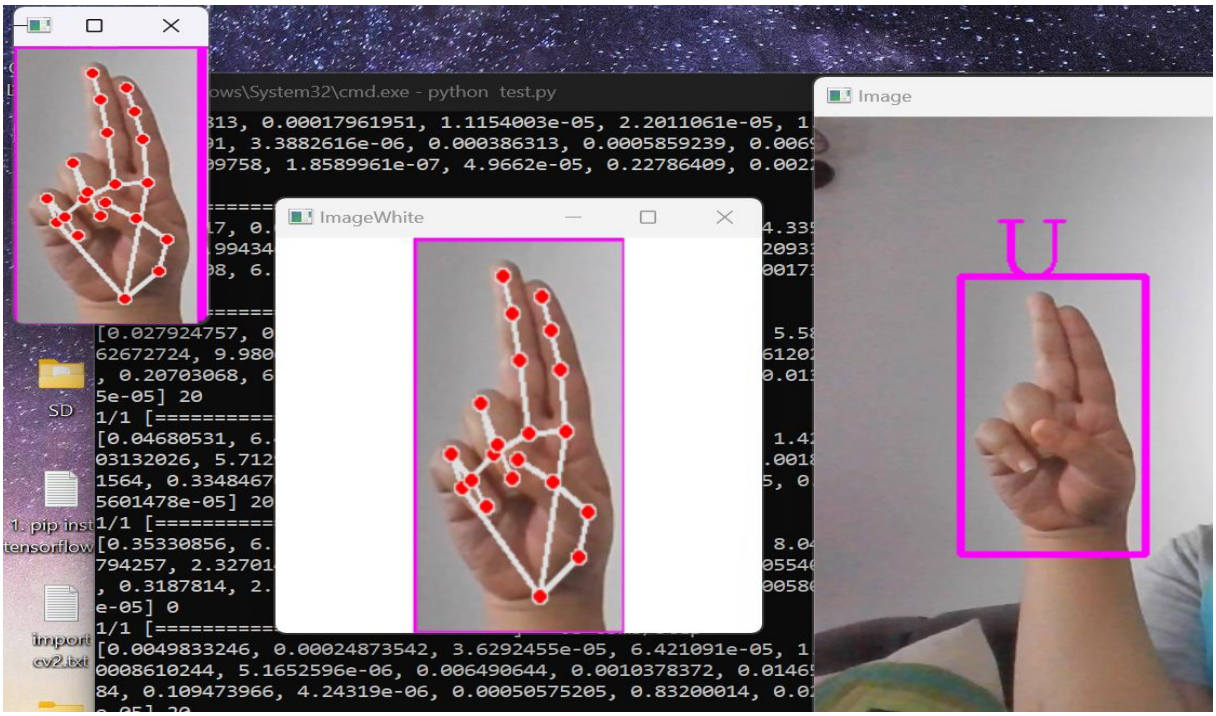
19.S


ImageWhite
Image
S
Ln 17, Col 140

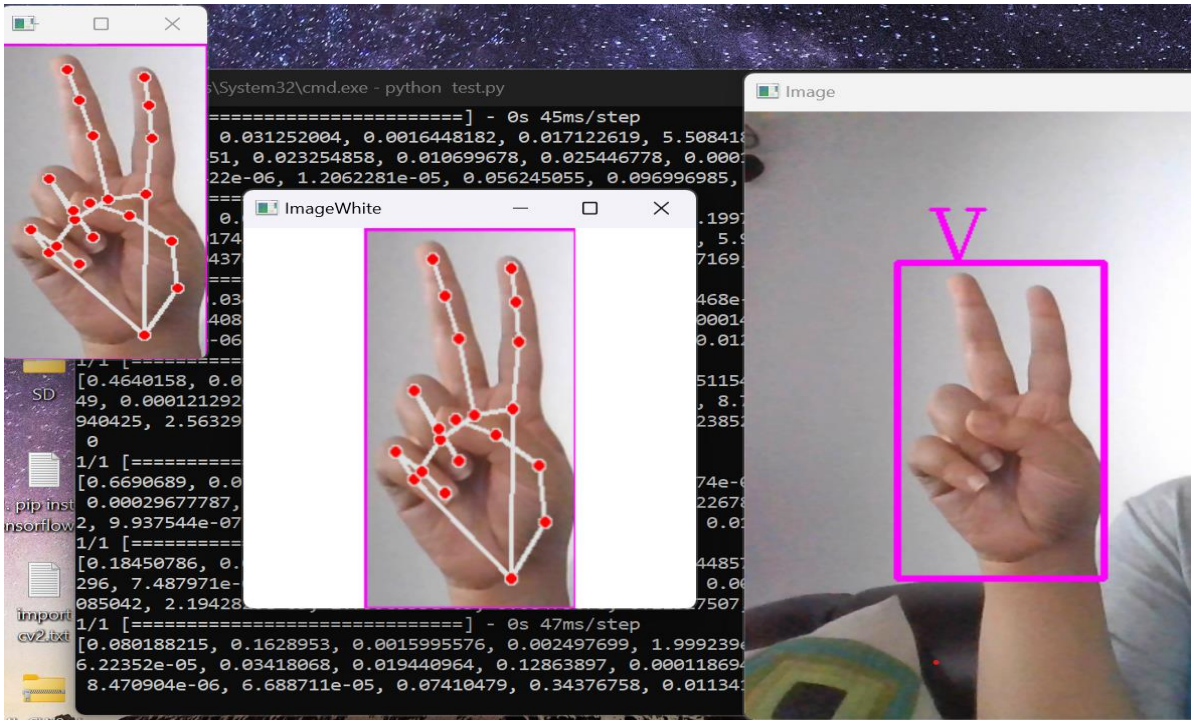
20.T


ImageWhite
Image
T
SD
import cv2.txt

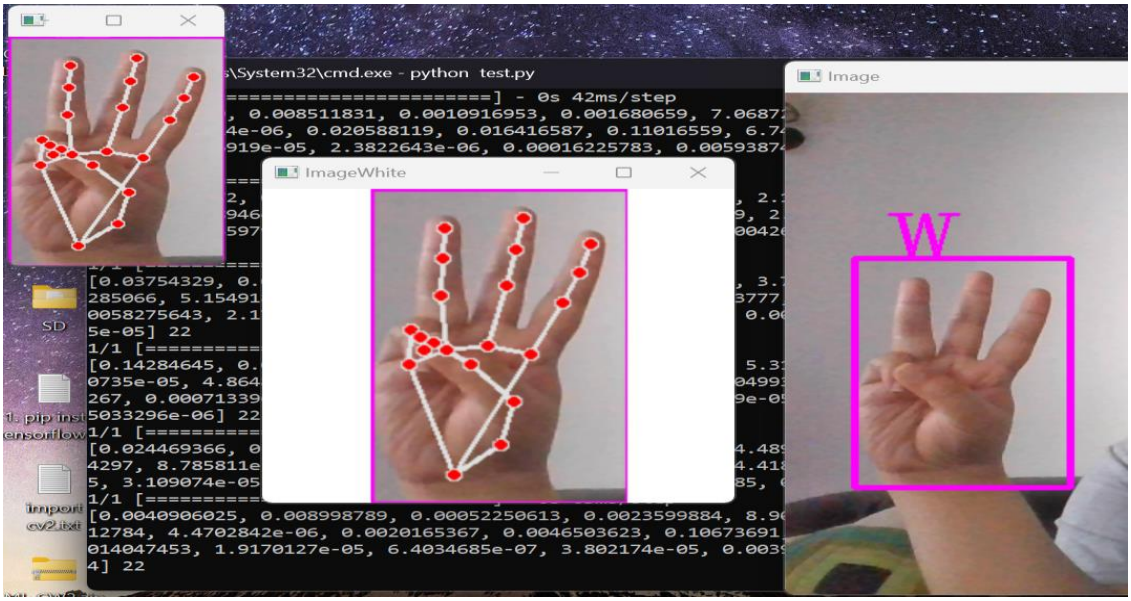
21.U



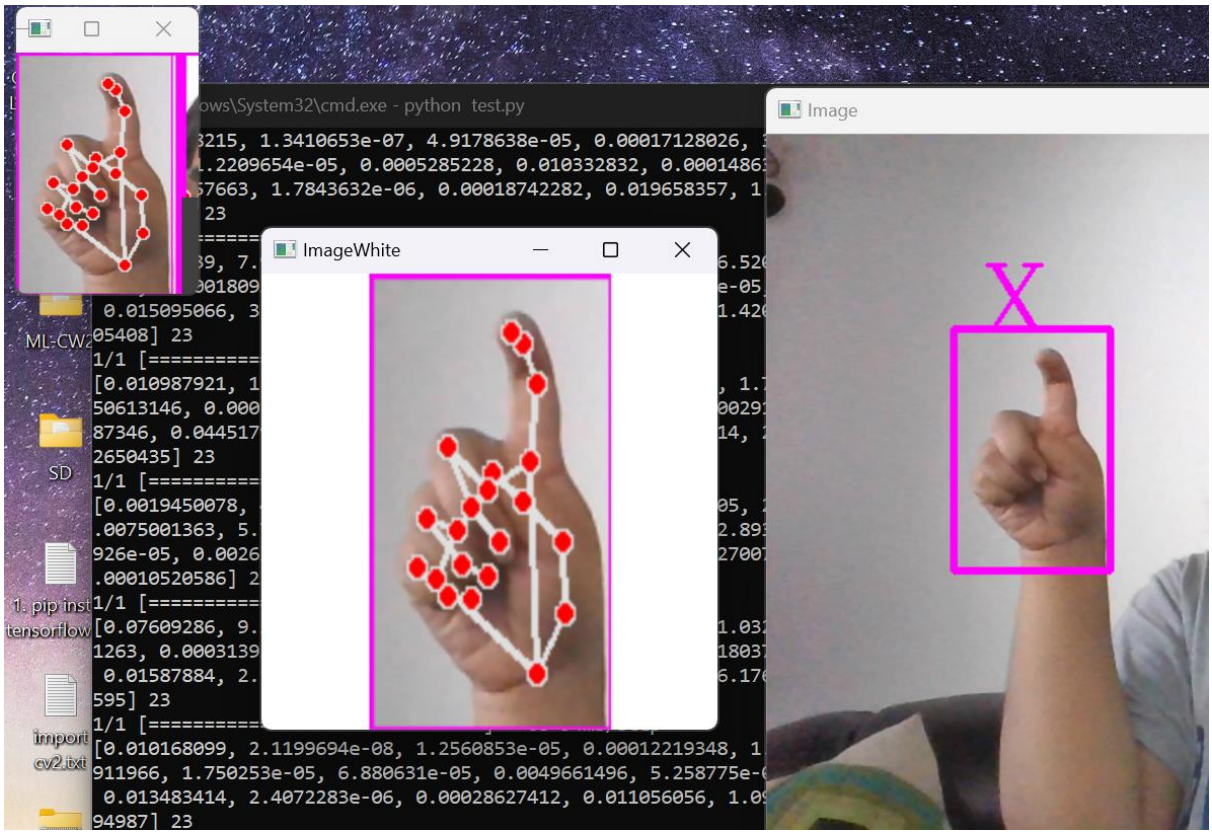
22.V



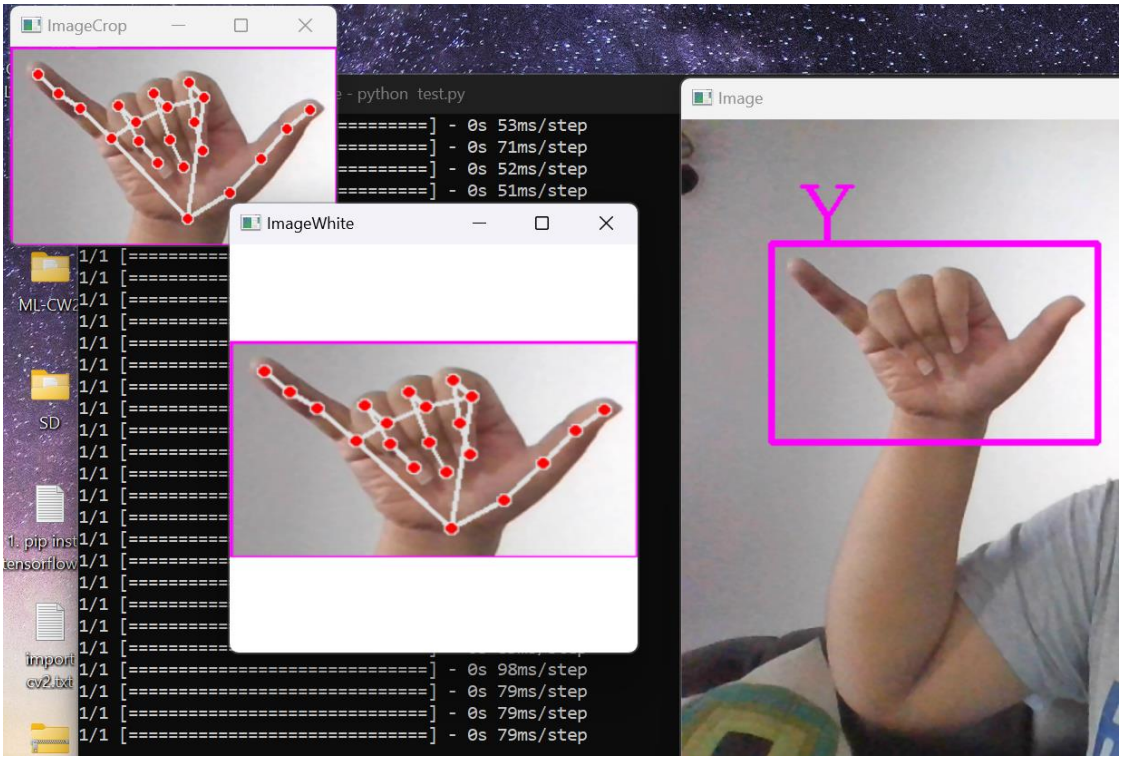
23.W


\System32\cmd.exe - python test.py
ImageWhite
Image
W

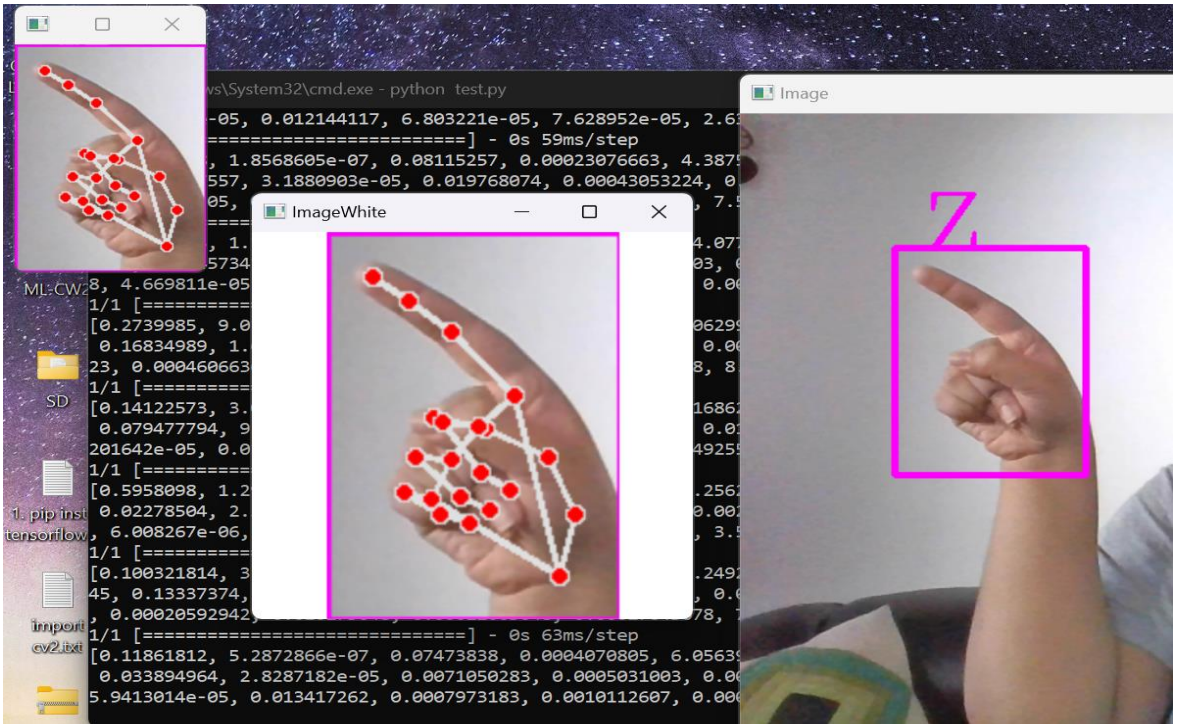
24.X


ows\System32\cmd.exe - python test.py
ImageWhite
Image
X

25.Y


ImageCrop
ImageWhite
Image
Y
- python test.py
] - 0s 53ms/step
] - 0s 71ms/step
] - 0s 52ms/step
] - 0s 51ms/step
] - 0s 98ms/step
] - 0s 79ms/step
] - 0s 79ms/step
] - 0s 79ms/step

26.Z


ImageWhite
Image
Z
] - 0s 59ms/step
] - 0s 63ms/step

4. SOURCE CODE

4.1. Training.py

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


# Initialize camera and hand detector

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


# Offset and size parameters for image processing

offset = 30
imgSize = 300


# Folder path to save images

folder = r"C:\Users\shazna salman\Desktop\ML-CW2\dataset\Z"
counter = 0


while True:

    # Read a frame from the camera

    success, img = cap.read()

    # Find hands in the frame
    hands, img = detector.findHands(img)

    if hands:
        # Take the first detected hand
        hand = hands[0]
        x, y, w, h = hand['bbox']

        # Create a white image matrix

        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
```

```
    # Crop the detected hand region
    imgCrop = img[y - offset : y + h + offset, x - offset : x + w + offset]

    # Get dimensions and aspect ratio of the cropped image
    imgCropShape = imgCrop.shape
    aspectRatio = h / w

    # Resize and fit the cropped image into the white matrix
    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

    # Display the cropped and white images
    cv2.imshow("ImageCrop", imgCrop)
    cv2.imshow("ImageWhite", imgWhite)

# Display the original frame
cv2.imshow("Image", img)

# Check for key press (save image when 'a' is pressed)
key = cv2.waitKey(1)
if key == ord("a"):
    counter += 1

    # Save the white image with a unique filename
    cv2.imwrite(f'{folder}/Image_{time.time()}.jpg', imgWhite)
    print(counter)
```

4.2. test.py

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

# Initialize camera, hand detector, and hand classifier
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
handClassifier = Classifier("ModelTrain/keras_model.h5", "ModelTrain/labels.txt")

# Offset and size parameters for image processing
offset = 20
imgSize = 300

# Folder path to save images (if required)
folder = r"C:\Users\shazna salman\Desktop\ML-CW2\dataset\A"
count = 0

# Labels for gesture recognition
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:
    # Read a frame from the camera
    success, img = cap.read()
```

```
# Create a copy of the original frame
imgOutput = img.copy()

# Find hands in the frame
hands, img = detector.findHands(img)

if hands:
    # Take the first detected hand
    hand = hands[0]
    x, y, w, h = hand['bbox']

    # Create a white image matrix
    imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255

    # Crop the detected hand region
    imgCrop = img[y - offset : y + h + offset, x - offset : x + w + offset]

    # Get dimensions and aspect ratio of the cropped image
    imgCropShape = imgCrop.shape
    aspectRatio = h / w

    # Resize and fit the cropped image into the white matrix
    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 = handClassifier.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 = handClassifier.getPrediction(imgWhite, draw=False)

        # Display the recognized label on the output frame
        cv2.putText(imgOutput, labels[index], (x, y - 20), cv2.FONT_HERSHEY_COMPLEX, 2, (255, 0, 255), 2)
        cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w + offset, y + h + offset), (255, 0, 255), 4)

        # Display the cropped and white images
        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("ImageWhite", imgWhite)

    # Display the original frame
    cv2.imshow("Image", imgOutput)

    # Check for key press (if required to save images)
    key = cv2.waitKey(1)
    if key == ord("a"):
        count += 1
        # Save the white image with a unique filename
        cv2.imwrite(f'{folder}/Image_{count}.jpg', imgWhite)
        print(count)
```

5. CONCLUSION

In conclusion, this Python script integrates computer vision techniques and deep learning to create a gesture recognition system. The program uses a webcam to capture real-time video frames, employs a hand detection module to locate and track the user's hand, and utilizes a pre-trained deep learning model for classifying American Sign Language (ASL) gestures corresponding to letters A-Z. The recognized letter is then displayed on the video feed, providing a practical demonstration of how technology can assist individuals who use sign language as their primary means of communication. This project enhances accessibility for the deaf and hard-of-hearing community by bridging communication gaps, showcasing the potential of technology to positively impact diverse user groups.

Real world applications

1. *Classrooms*: Real-time sign language interpretation for deaf or hard-of-hearing students during lectures and discussions.
2. *Online Learning:* Integration into virtual learning environments for remote education.
3. *Service Desks*: Improving communication at customer service desks for customers who use sign language.
4. *Call Centers*: Enhancing accessibility during customer support calls.
5. *Mobile Applications:* Integration into mobile apps for on-the-go sign language interpretation.
6. *Medical Settings:* Facilitating communication between healthcare professionals and patients with hearing impairments.
7. *Telehealth:* Bridging communication gaps in remote healthcare consultations.
8. *Conferences and Presentations:* Providing sign language interpretation for attendees with hearing impairments.
9. *Cultural Events:* Enhancing inclusivity in cultural events.
10. *Smart Assistants:* Enhancing accessibility of voice-activated devices for sign language users.

11. *Video Calls:* Real-time sign language interpretation in video calls on social media and communication apps.
12. *Call Centers*: Supporting effective communication during emergency calls.

6. REFERENCES

1. **To create the model**

Google. (2023) 'Teachable Machine.' [online] Google. Available at: https://teachablemachine.withgoogle.com/train/image (Accessed: 16 December 2023)

2. **Research ASL datasets (Kaggle)**

Kaggle. (2023) 'Kaggle: Your Machine Learning and Data Science Community.' [online] Kaggle. Available at: https://www.kaggle.com/ (Accessed: 16 December 2023)

3. **Videos**
   - Murtaza (2022) 'Easy Hand sign Detection- Computer Vision' [online] YouTube. Available at: https://www.youtube.com/watch?v=wa2ARoUUdU8 (Accessed: 16 December 2023)
   - Rennotte.N (2021) 'Sign Language Detection with Tensorflow'[online] YouTube. Available at: https://www.youtube.com/watch?v=wa2ARoUUdU8 (Accessed: 16 December 2023)

4. **Diagram used for ASL**