Bachelor of Engineering
Mini-Project Presentation

Department of Electronics and Computer Science
Shree L R Tiwari College of Engineering
Kanakia Park, Mira Road (E), Mumbai-401107
UNIVERSITY OF MUMBAI
2023-2024

SIGN LANGUAGE RECOGNITION FOR MUTE AND DEAF

MINI-PROJECT

Team Members

1st Member

Name: Darsh Jobanputra
Roll No.: 24
TE-ECS

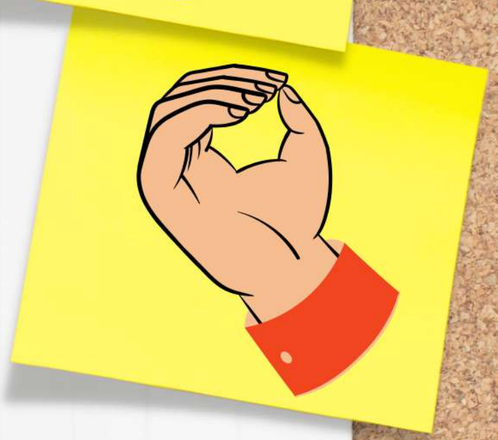
TE-ECS
Roll No.: 33
Name: Aarya Mourya

2nd Member

3rd Member

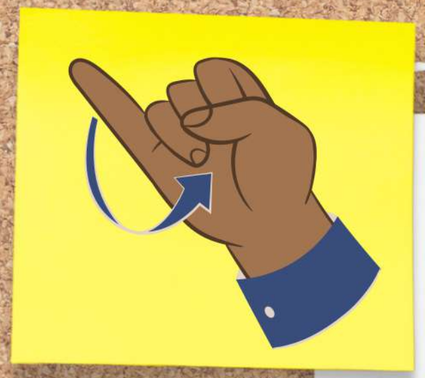
Name: Vatsal Nandawana
Roll No.: 34
TE-ECS

Contents



Introduction

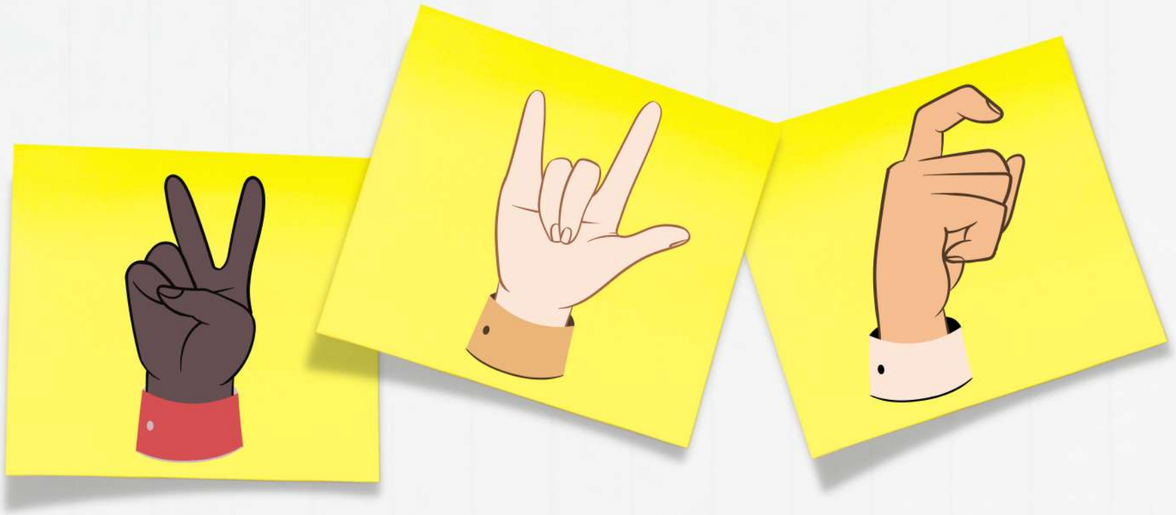
- American Sign Language (ASL) is crucial for communication among Deaf and Dumb (D&M) individuals.
- Sign language involves gestures, facial expressions, and body movements.
- Sign language translation, particularly fingerspelling recognition, is essential to bridge the communication gap.
- The project focuses on creating a model to recognize fingerspelling gestures, enabling accurate conversion of ASL into written or spoken language.

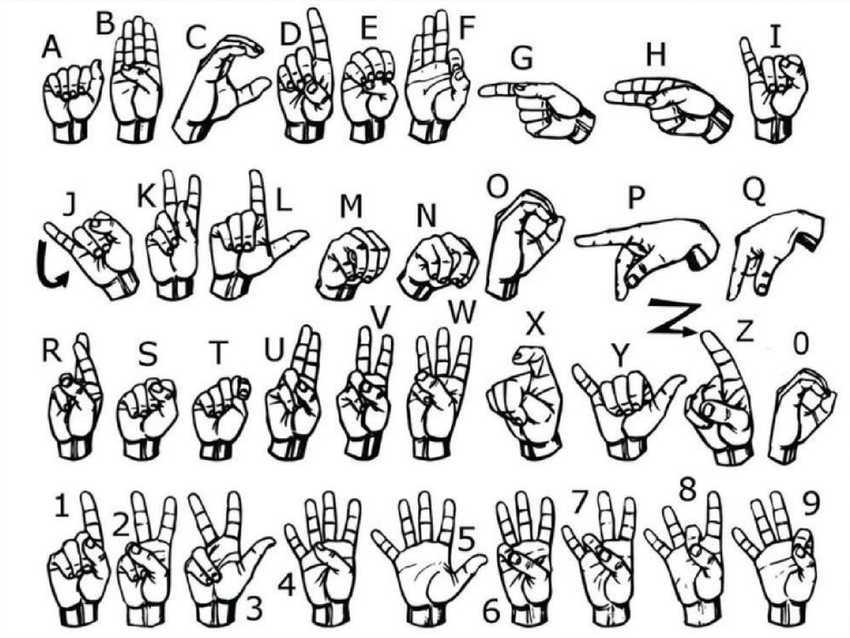


Abstract

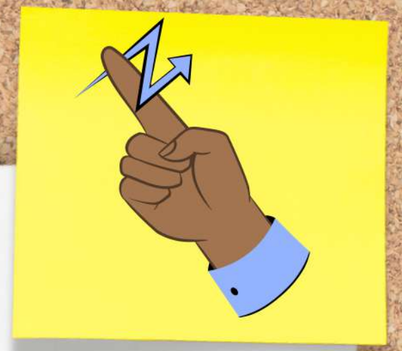
- A computer vision-based sign language recognition system using Machine Learning models, Python, and OpenCV.
- The system accurately recognizes American Sign Language (ASL) gestures in real time, enabling efficient communication for newly deaf/mute individuals.
- By training the model on a comprehensive dataset, we ensure rapid and precise conversion of ASL gestures into text, promoting inclusivity.
- Thorough testing proves the system works well in different situations, connecting people with and without disabilities and promoting an inclusive society.

Communication Element	Description
Fingerspelling	Used to spell words letter by letter.
Word Level Sign Vocabulary	Sign language vocabulary at the word level.
Non-Manual Features	Involves facial expressions, tongue, mouth, and body position.





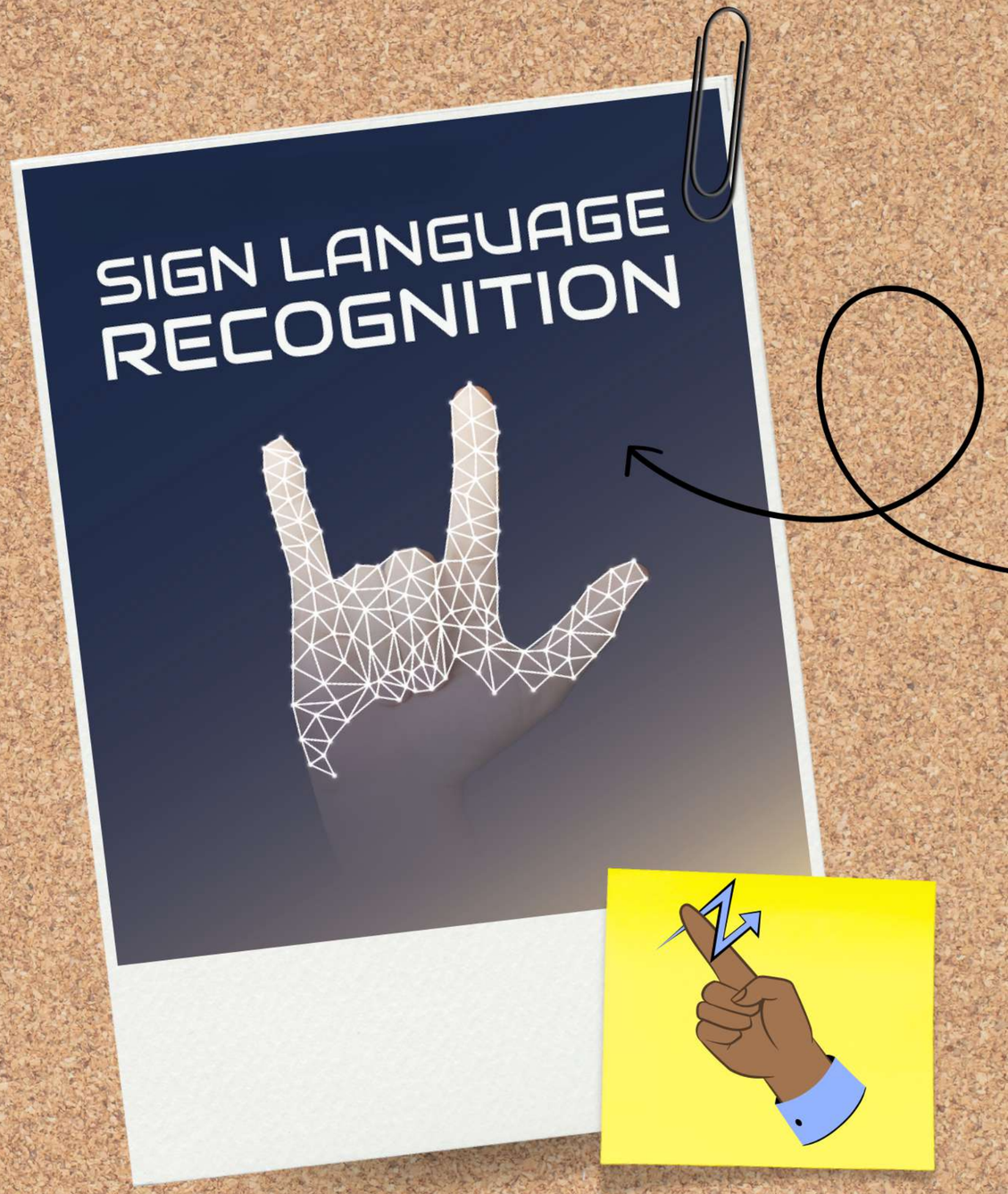
Literature Survery

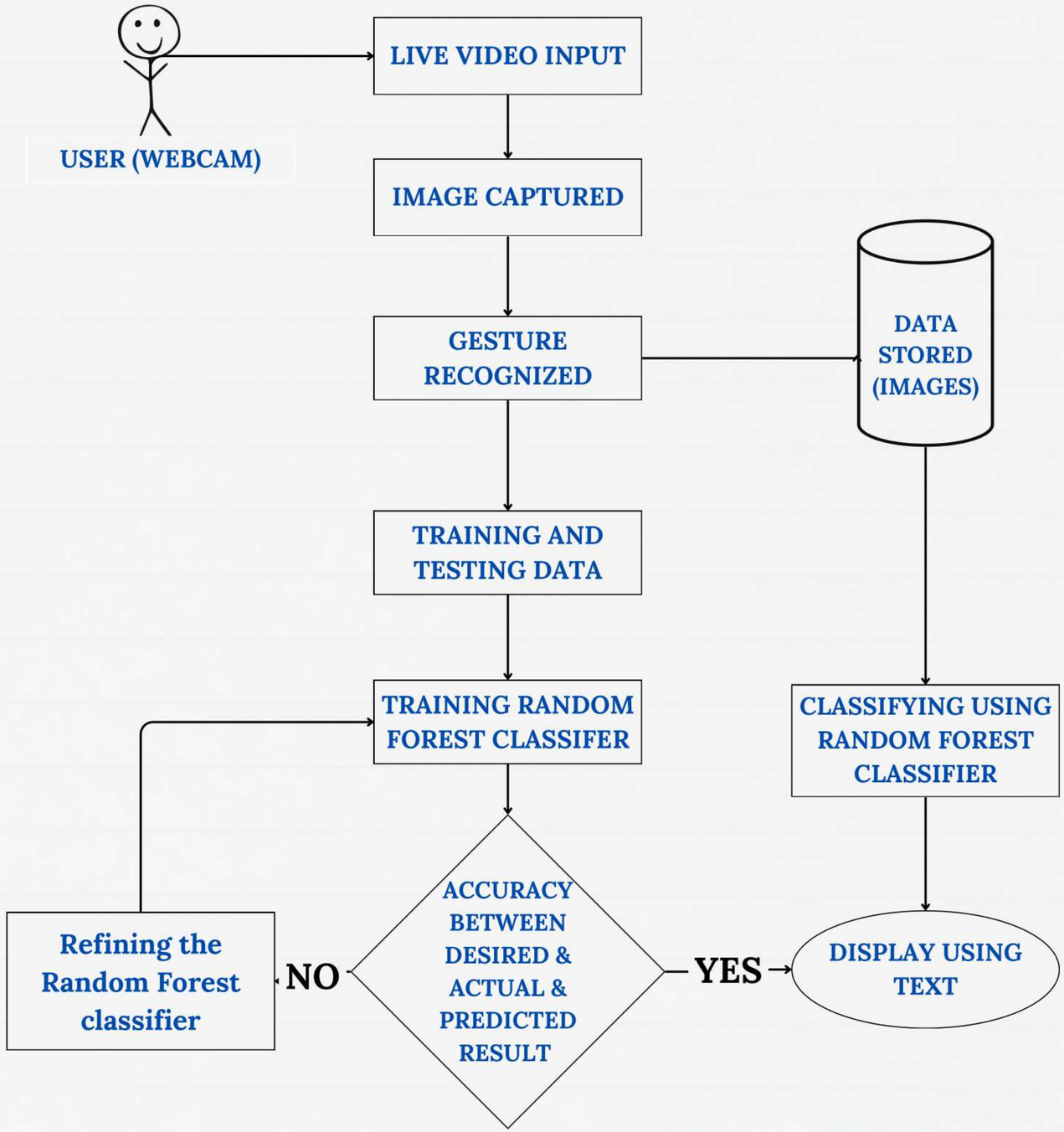


- "Sign Language Recognition with Unsupervised Feature Learning" by Graham Taylor, Geoffrey Hinton, and Sam Roweis.
This paper explores unsupervised feature learning techniques for sign language recognition.
- "Sign Language Recognition: A Comparative Study of the State of the Art" by Sergio Montazzolli Silveira, Claudio Rosito Jung, and Anderson Rocha. This paper provides a comparative analysis of different techniques used in sign language recognition.
- "Sign Language Recognition Using 3D Convolutional Neural Networks" by Long Duong, Cuong Nguyen, et al. This paper focuses on using 3D CNNs for sign language recognition, considering the temporal aspects of sign language gestures.
- "Sign Language Recognition Based on 3D Convolutional Neural Networks" by Zhenyang Li, Sheng Huang, et al. This paper explores 3D CNNs and their application in sign language recognition, emphasizing the importance of capturing spatiotemporal information.
- "Sign Language Recognition Using Principal Component Analysis and Neural Networks" by Thirimachos Bourlai and Ioannis Pavlidis. This paper discusses the use of Principal Component Analysis (PCA) combined with neural networks for sign language recognition.

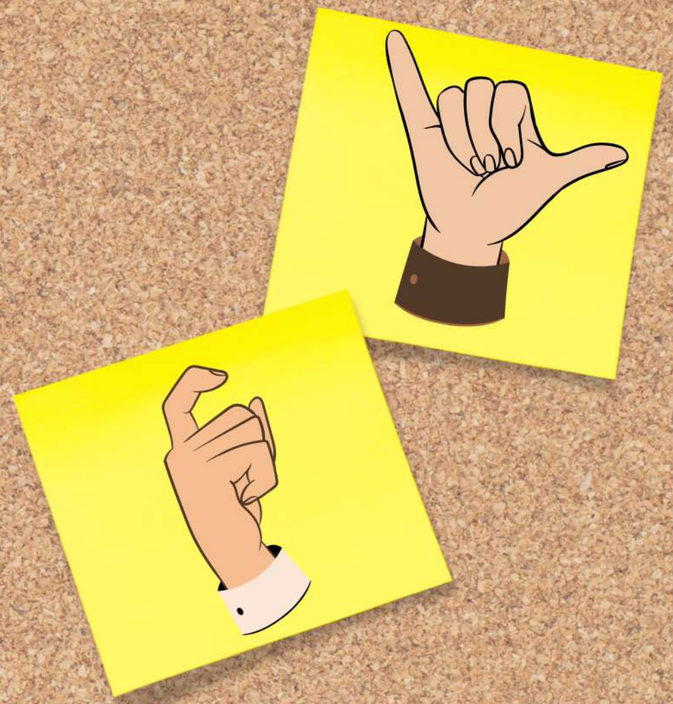
Overview

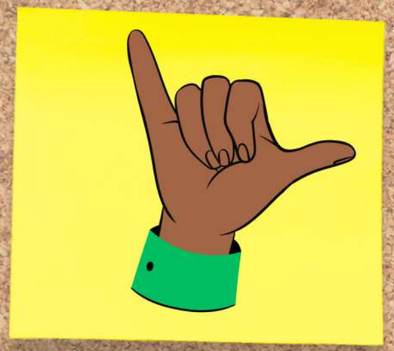
- The project employs Python and OpenCV to develop a real-time sign language recognition system using computer vision and Machine Learning.
- It understands American Sign Language, helping the deaf communicate and fostering inclusivity




USER (WEBCAM)
LIVE VIDEO INPUT
IMAGE CAPTURED
GESTURE RECOGNIZED
DATA STORED (IMAGES)
TRAINING AND TESTING DATA
TRAINING RANDOM FOREST CLASSIFER
CLASSIFYING USING RANDOM FOREST CLASSIFIER
ACCURACY BETWEEN DESIRED & ACTUAL & PREDICTED RESULT
NO
YES
Refining the Random Forest classifier
DISPLAY USING TEXT

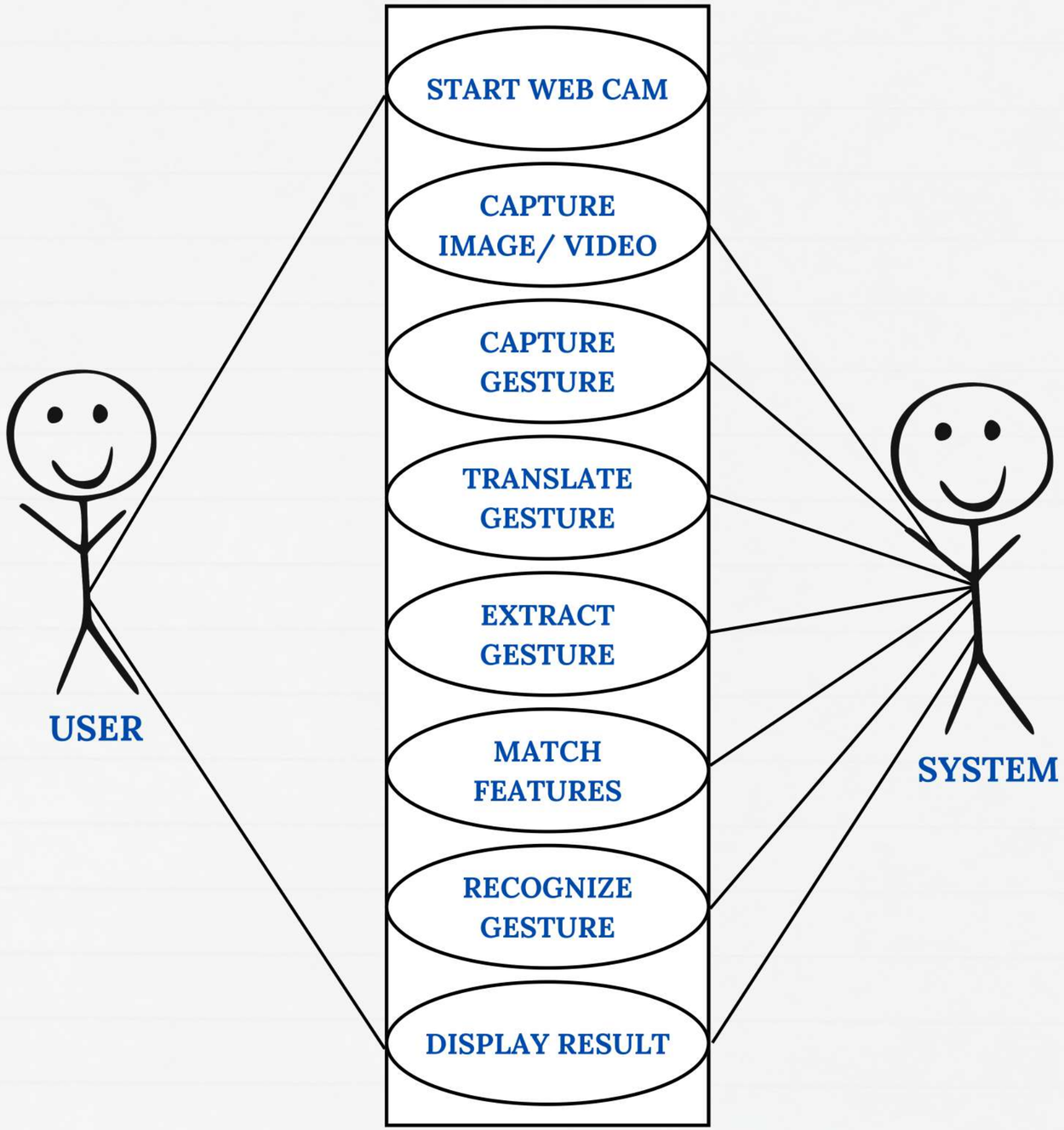
Flowchart





PROCESS

1. **Image Acquisition Signs**
   - American Sign Language (ASL) symbols
   - **Recorded gestures for each sign symbol**
   - **(e.g., A, B, L)**
2. **Image Acquisition Device**
   - COMPUTER
   - WEBCAM
3. **Data Collection**
   - Capturing sign language gestures for training and testing.
   - Record data using a webcam.
   - 100 images per alphabet.
   - Organize data by class (folders).
4. **Data Preprocessing (Image Preprocessing)**
   - Preparing data for landmark detection and feature extraction.
   - Extract picture frames.
   - Pre-process frames (resize, etc.).
   - Background subtraction (optional).
   - Noise reduction (optional).
5. **Hand Detection and Training**
   - Extract frames from recorded images.
   - Pre-process frames (resize, normalize).
   - Landmark detection for hand position extraction.
6. **Feature Extraction**
   - Extracting relevant features from landmarks or segmented hand images for model training.
   - -Landmark-based: - Relative landmark positions.
   - -Inter-landmark distances.
   - -Angles between landmarks.
   - -Hand landmarks (x,y,z coordinates).
7. **Model Training (Classifier Learning)**
   - -Label assignment (based on folder).
   - -Split data into training and testing sets.
   - -Train a Random Forest classifier using Scikit Learn.
   - Training the Random Forest classifier for sign recognition.
8. **Training Data Set & Model Evaluation**
   - Data used for model training.
   - - Hand landmark data (or segmented hand images).
   - - Labels (corresponding sign symbols).
   - -Assessing the performance of the trained model on unseen data.
9. **Landmark Detection (Real-time)**
   - Identifying hand landmarks in live video frames for feature extraction.
   - -Utilize a pre-trained MediaPipe model.
   - -Detect hand landmarks (x,y,z coordinates) in real-time.
10. **Classification and Recognition**
    - Using the trained model to recognize signs from live data.
    - - Test the trained model using live webcam data.
    - - Utilize MediaPipe for hand landmark detection.
    - - Classify hand poses using the trained model.
    - - Draw bounding boxes and labels on the webcam feed.
11. **TEXT OUTPUT**



Usecase

USE CASE SCENARIO FOR SIGN LANGUAGE RECOGNITION SYSTEM	
Usecase name	Sign Language Recognition
Participating Actores	User, System
Flow of Events	Start The System(U) Capturing Videos(S) Translate Gesture(S) Extract Feature(S) Match Features(S) Recognizing Gestures(S) Display Result
Entry Condition	Run The Code
Exit Condition	Displaying The Label
Quality Requirements	Cam Pixels Clarity, Good Light Condition

Matplotlib

Matplotlib is a Python library for creating static, interactive, and animated visualizations. It is used in the project for plotting the image onto a 2D x, and y-axis so that each pixel of the image is associated with a coordinate pair for visualizing data, gesture recognition, and training process.

Scikit-learn:

Scikit-learn is a Python library for machine learning tasks. In this project, it will be used later to build accurate sign language recognition models. Its simplicity and various algorithms make it ideal for efficient communication solutions for the hearing -impaired.

OpenCV:

OpenCV, an open-source computer vision library, provides essential tools for image processing, feature extraction, and gesture recognition. It offers functions to manipulate images and extract relevant features from sign language gestures.

Linux:

Linux, as an open-source operating system, provides a stable and secure environment for your project. It allows efficient resource utilization and supports various programming tools and libraries essential for developing complex applications like sign language recognition.

Python:

Data Collection:
Gather a dataset of sign language gestures. Ensure the dataset covers various gestures and is well-labeled.

Data Preprocessing:
Clean and preprocess the dataset.

Feature Extraction:
Extract relevant features from the images.

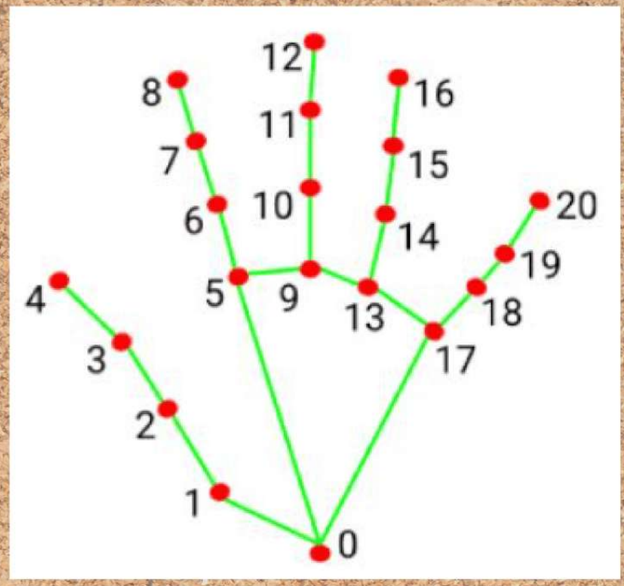
Model Selection and Training:
Choose an appropriate machine learning model. Train the selected model using your preprocessed dataset.

Computer Vision

Utilizing computer vision techniques, your project processes visual data (sign language gestures) to extract meaningful information. Computer vision algorithms help in understanding and interpreting gestures accurately.

MediaPipe HandPose:

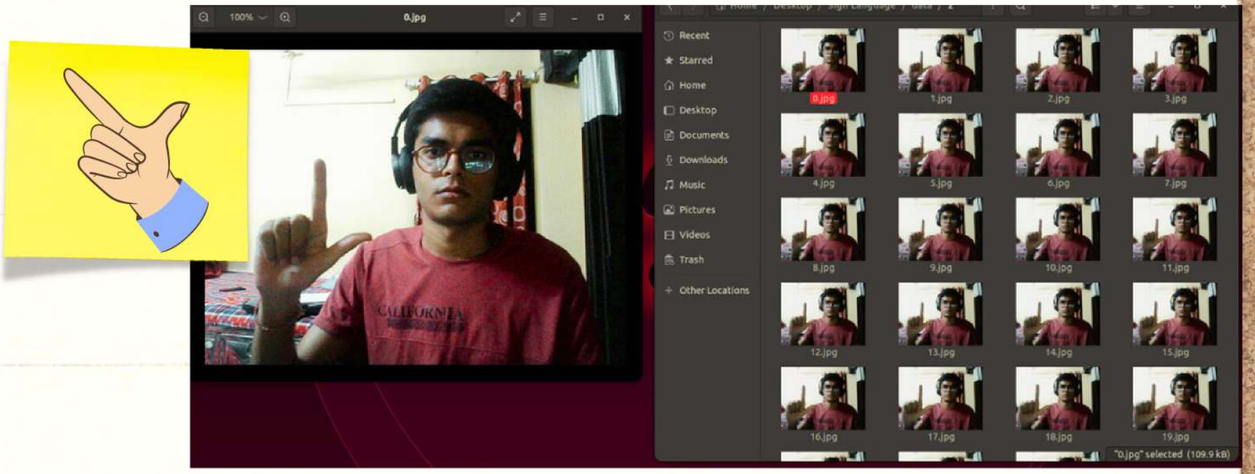
MediaPipe is a framework developed by Google that includes a hand landmark detection model called HandPose. It's based on a combination of convolutional neural networks (CNNs) and geometric constraints to accurately locate hand landmarks.

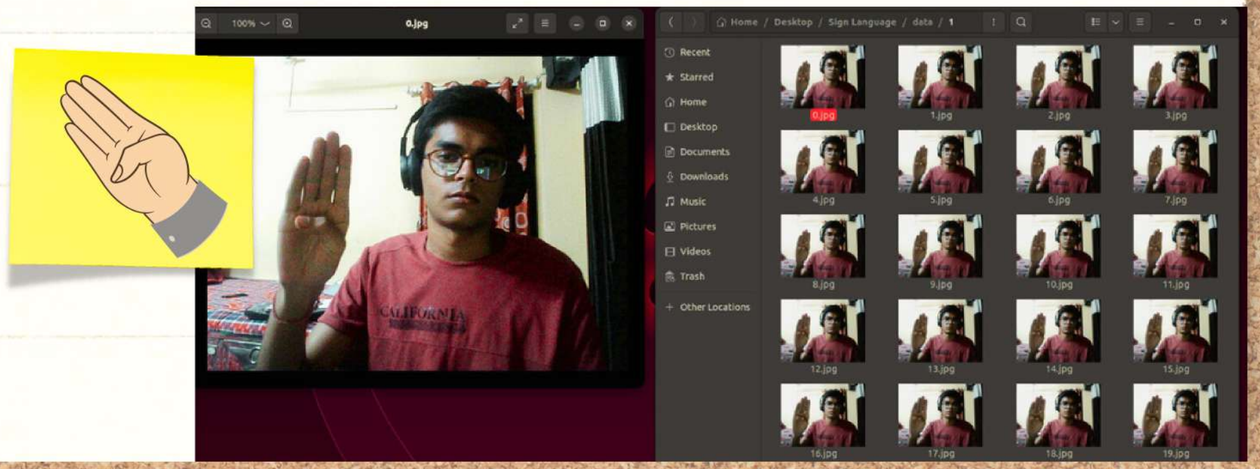


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

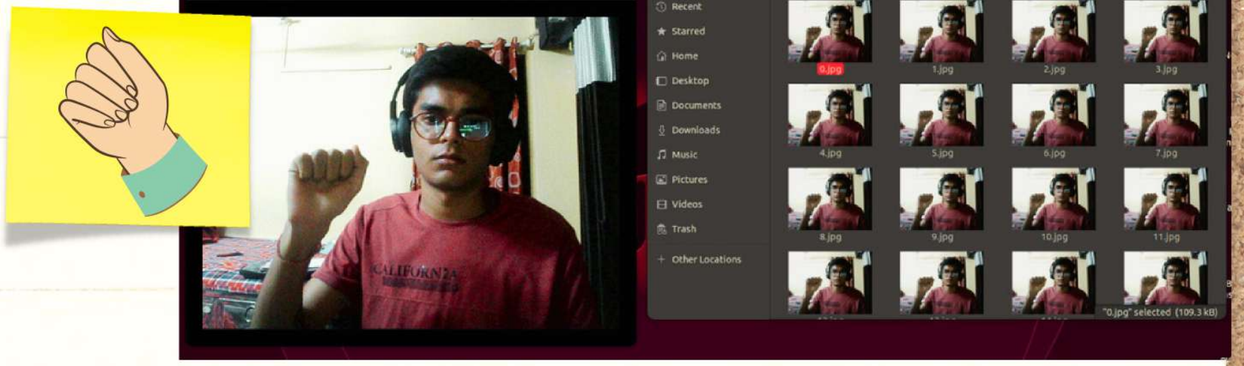
collect_imgs.py:

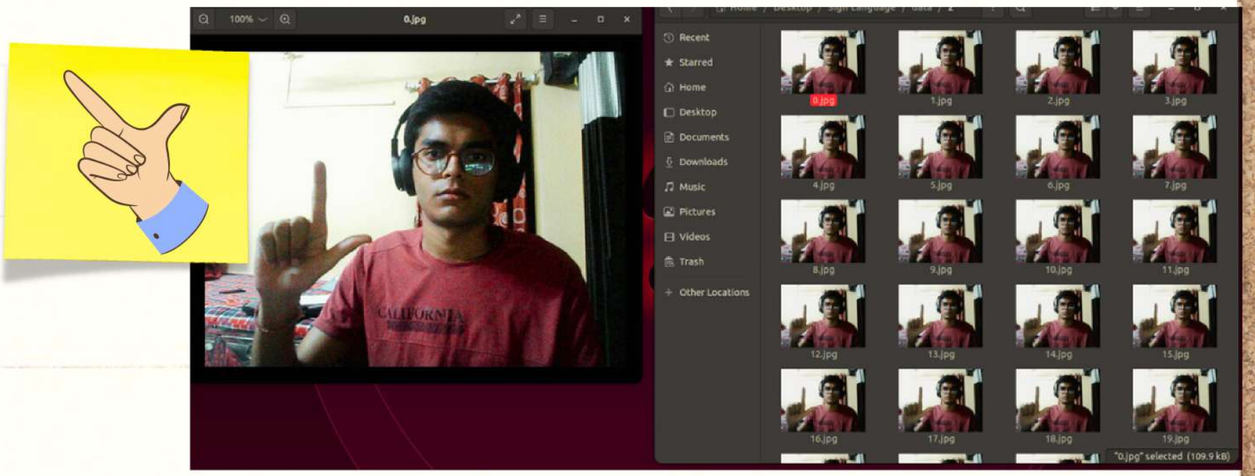
```
import os
import cv2
DATA_DIR = './data'
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)
number_of_classes = 3
dataset_size = 100
cap = cv2.VideoCapture(0) #the appropriate
index of cam device (/dev/video0)
for j in range(number_of_classes):
    if not os.path.exists(os.path.join(DATA_DIR,
str(j))):
        os.makedirs(os.path.join(DATA_DIR, str(j)))
    print('Collecting data for class {}'.format(j))
cv2.destroyAllWindows()
```

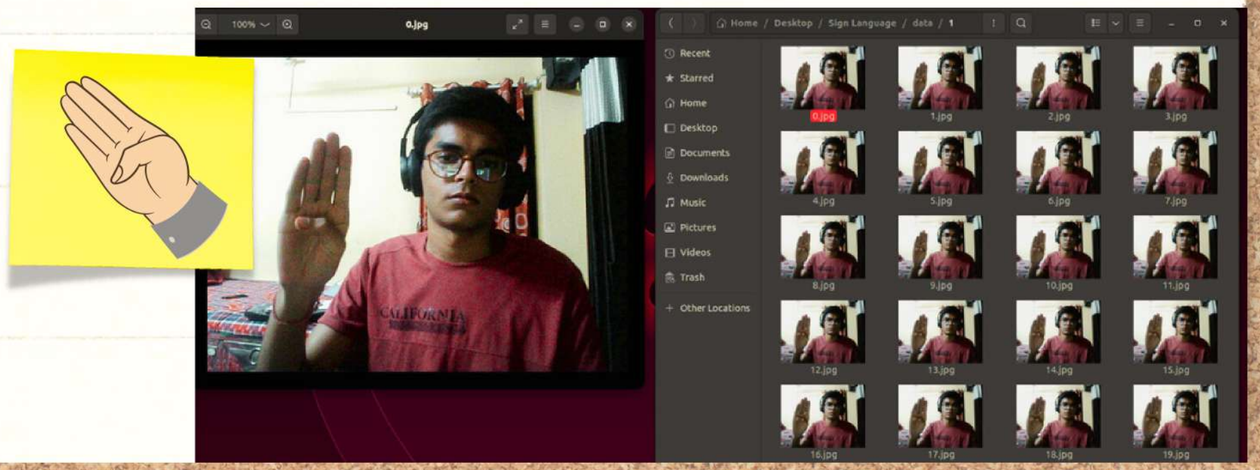




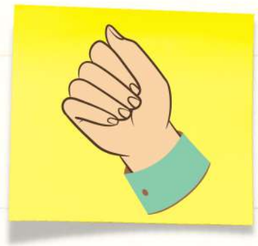
```
done = False
 while True:
 ret, frame = cap.read()
 cv2.putText(frame, 'Ready? Press "Q" ! :)', (100, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 255, 0), 3,
 cv2.LINE_AA)
 cv2.imshow('frame', frame)
 if cv2.waitKey(25) == ord('q'):
 break
 counter = 0
 while counter < dataset_size:
 ret, frame = cap.read()
 cv2.imshow('frame', frame)
 cv2.waitKey(25)
 cv2.imwrite(os.path.join(DATA_DIR, str(j),
'{}.jpg'.format(counter)), frame)
 counter += 1
cap.release()
```

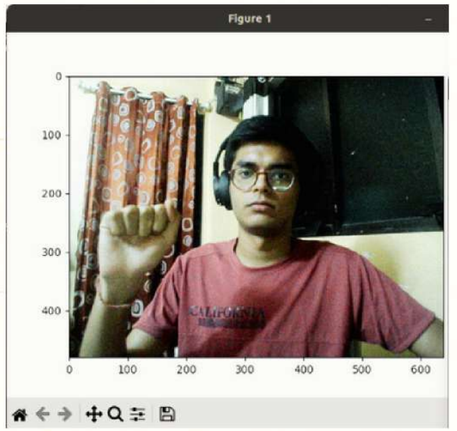


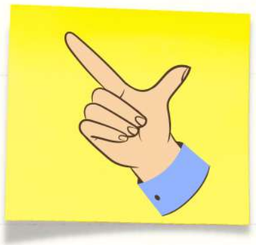


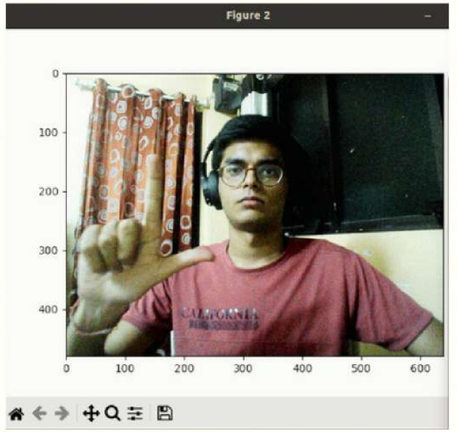


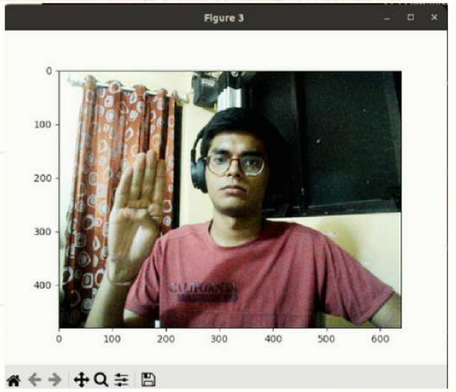
```
test_create_dataset.py:
import os import pickle import cv2
import matplotlib.pyplot as plt DATA_DIR = './data'
# Ensure DATA_DIR exists and is a directory if not
os.path.isdir(DATA_DIR):
print(f"Error: {DATA_DIR} is not a directory.") exit()
for dir_ in os.listdir(DATA_DIR):
dir_path = os.path.join(DATA_DIR, dir_) if
os.path.isdir(dir_path):
for img_path in os.listdir(dir_path)[:1]:
img = cv2.imread(os.path.join(dir_path, img_path)) img_rgb =
cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.figure() plt.imshow(img_rgb)
plt.show()
```





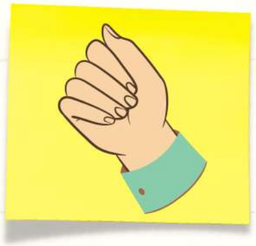




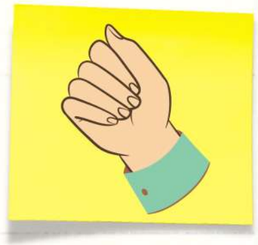


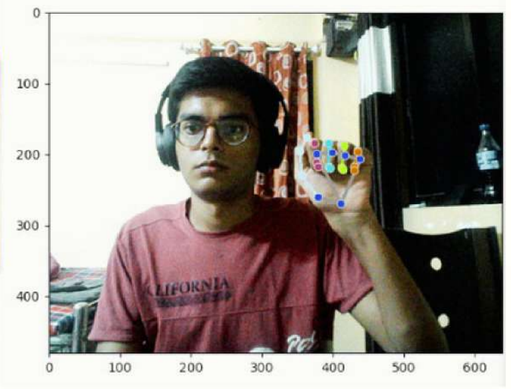
test_detect_landmark.py:

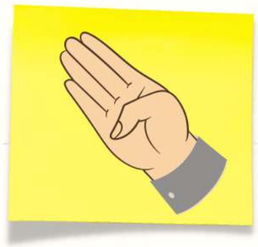
```
import os
import pickle
import mediapipe as mp
import cv2
import matplotlib.pyplot as plt
mp_hands = mp.solutions.hands mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
hands = mp_hands.Hands(static_image_mode=True,
min_detection_confidence=0.1)
DATA_DIR = './data'
# Ensure DATA_DIR exists and is a directory
if not os.path.isdir(DATA_DIR):
print(f"Error: {DATA_DIR} is not a directory.")
exit()
for dir_ in os.listdir(DATA_DIR):
dir_path = os.path.join(DATA_DIR, dir_) if os.path.isdir(dir_path):
for img_path in os.listdir(dir_path)[:1]:
```

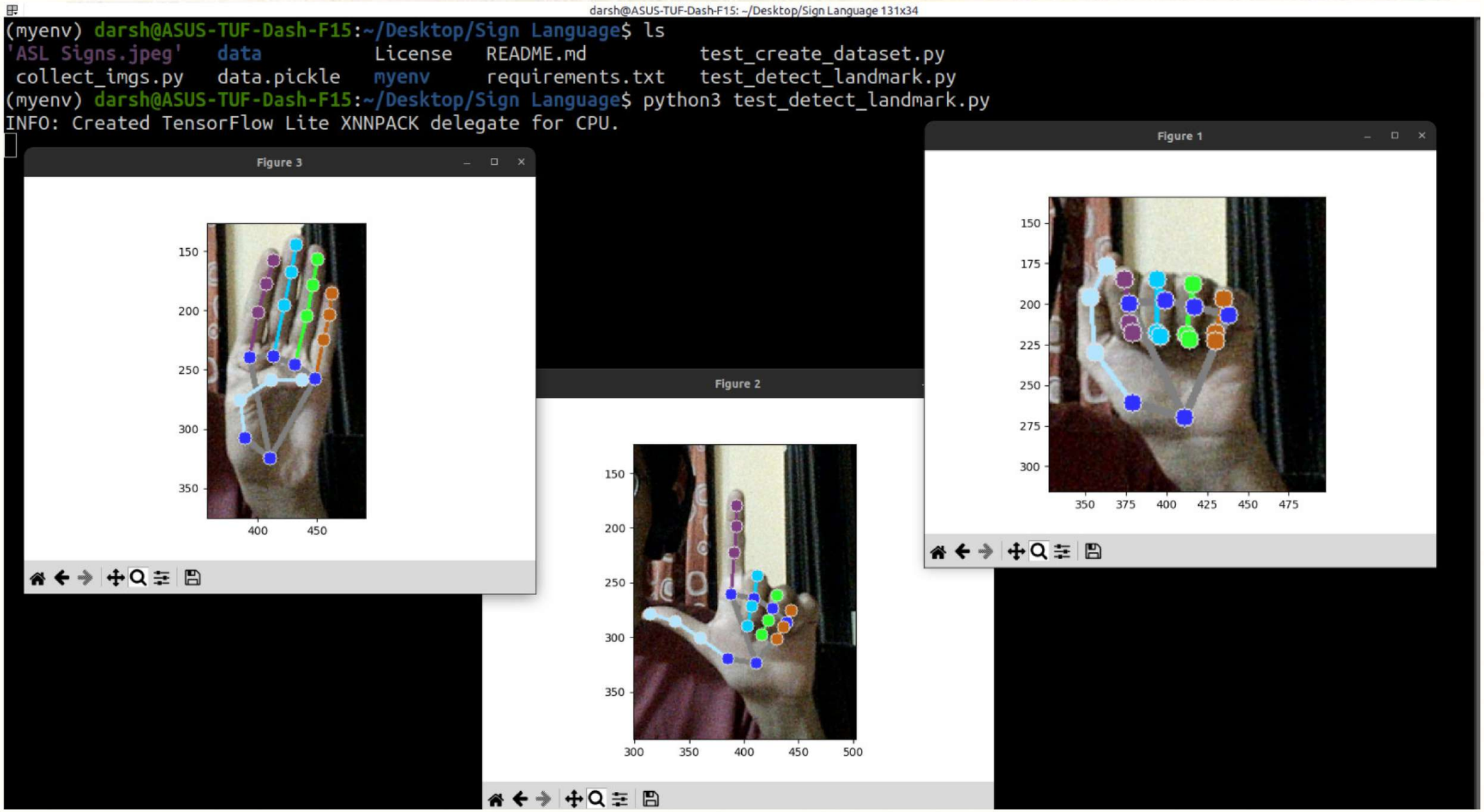


```
img = cv2.imread(os.path.join(dir_path, img_path)) img_rgb =
cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
results = hands.process(img_rgb) if
results.multi_hand_landmarks:
for hand_landmarks in results.multi_hand_landmarks:
mp_drawing.draw_landmarks(
img_rgb, hand_landmarks,
mp_hands.HAND_CONNECTIONS,
mp_drawing_styles.get_default_hand_landmarks_style(),
mp_drawing_styles.get_default_hand_connections_style())
plt.figure() plt.imshow(img_rgb)
plt.show()
```









Hand Landmarks Detected (Zoomed Version)

create_dataset.py [CO-ORDINATES of Hand Landmarks printed]

- **TensorFlow Lite XNNPACK Delegate:**

This line indicates that a TensorFlow Lite delegate called XNNPACK has been created for CPU inference acceleration.

- **XYZ Coordinates of Hand Landmarks:**

For each hand landmark, XYZ coordinates are provided. These coordinates represent the spatial position of each landmark relative to the camera.

Each set of coordinates consists of:

x: The X-coordinate.

y: The Y-coordinate.

z: The Z-coordinate.

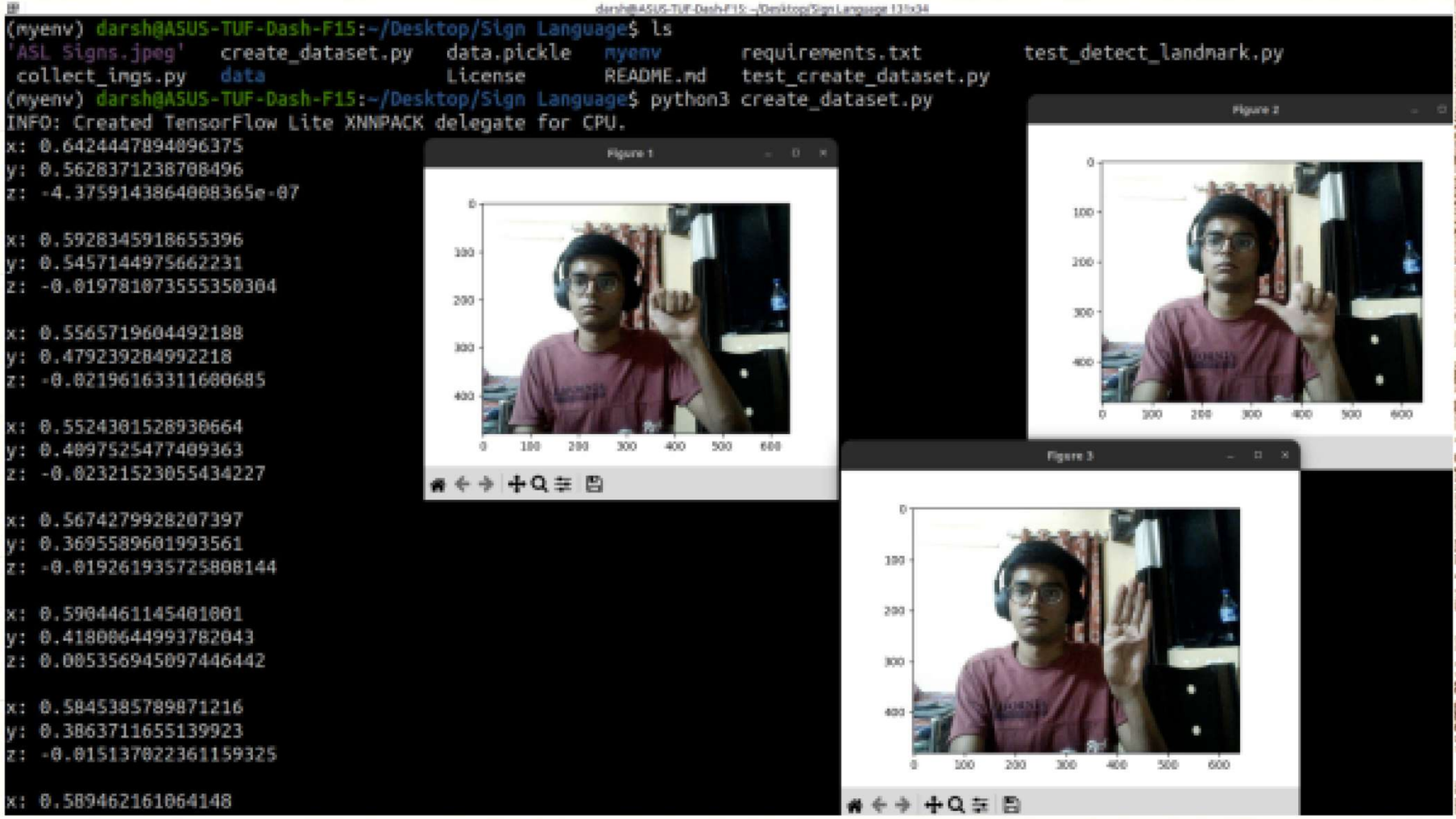


Figure1,2,3 - Numerical values are arranged in a visual representation, possibly representing axes or dimensions.

temp.txt | test_create_dataset.py | test_detect_landmark.py | create_dataset.py

```
14
15 DATA_DIR = './data'
16
17 # Ensure DATA_DIR exists and is a directory
18 if not os.path.isdir(DATA_DIR):
19     print(f"Error: {DATA_DIR} is not a directory.")
20     exit()
21
22 data = []
23 labels = []
24 for dir_ in os.listdir(DATA_DIR):
25     dir_path = os.path.join(DATA_DIR, dir_)
26     if os.path.isdir(dir_path):
27         for img_path in os.listdir(dir_path):
28             data_aux = []
29             img = cv2.imread(os.path.join(dir_path, img_path))
30             img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
31
32             results = hands.process(img_rgb) #code from here used
   for detecting hand landmarks
33             if results.multi_hand_landmarks:
34                 for hand_landmarks in results.multi_hand_landmarks:
35                     for i in
   range(len(hand_landmarks.landmark)):
36                         x = hand_landmarks.landmark[i].x
37                         y = hand_landmarks.landmark[i].y
38                         data_aux.append(x)
39                         data_aux.append(y)
40
41
42             data.append(data_aux)
43             labels.append(dir_)
44
45 f = open('data.pickle', 'wb')
46 pickle.dump({'data': data, 'labels':labels}, f)
47 f.close()
```

Python 2 | Tab Width: 8 | Ln 47, Col 10 | INS

darsh@ASUS-TUF-Dash-F15: ~/Desktop/Sign Language

darsh@ASUS-TUF-Dash-F15: ~/Desktop/Sign Language 90x20

```
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$ ls
'ASL Signs.jpeg'     data          myenv              test_create_dataset.py
 collect_imgs.py     data.pickle   README.md          test_detect_landmark.py
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$ python3 create_dataset.py
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$
```

Berryminator_
Task_Files-mas...

Home / Desktop / Sign Language

- Recent
- Starred
- Home
- Desktop
- Documents
- Downloads
- Music
- Pictures
- Videos
- Trash
- Other Locations

data
myenv
data.pickle
ASL Signs.jpeg
README.md
License
requirements.txt
collect_imgs.py
create_dataset.py
test_create_dataset.py
test_detect_landmark.py

temp.txt | test_create_dataset.py | test_detect_landmark.py | create_dataset.py | train_classifier.py

```python
import pickle

data_dict = pickle.load(open('./data.pickle', 'rb'))

print(data_dict.keys())
print(data_dict)
```

darsh@ASUS-TUF-Dash-F15: ~/Desktop/Sign Language 91x6

```
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$ ls
'ASL Signs.jpeg'     data          myenv              test_create_dataset.py
 collect_imgs.py     data.pickle   README.md          test_detect_landmark.py
 create_dataset.py   License       requirements.txt   train_classifier.py
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$ python3 train_classifier.py
```

darsh@ASUS-TUF-Dash-F15: ~/Desktop/Sign Language 82x36

```
73407, 0.6874133348464966, 0.4319711923599243, 0.6906089186668396, 0.366843879222
699, 0.6936632394790649, 0.3126072287559509, 0.7109906077384949, 0.551095843315124
5, 0.7192547917366028, 0.47811293601989746, 0.7211589813232422, 0.428387165069580
, 0.7215993404388428, 0.3848966956138611], [0.6450440883636475, 0.686363458633422
, 0.609196126461029, 0.6481707692146301, 0.6024258732795715, 0.5786869525909424,
.6458269953727722, 0.5400375723838806, 0.6916534304618835, 0.5373304486274719, 0.
140334010124207, 0.5015311241149902, 0.6234340071678162, 0.42010262608528137, 0.6
18886876106262, 0.36687085032463074, 0.6399289965629578, 0.32168927788734436, 0.6
72622752189636, 0.49659454822540283, 0.6593236923217773, 0.4053407907485962, 0.66
0631637573242, 0.3440757989883423, 0.6742997169494629, 0.29117757081985474, 0.676
629219055176, 0.5093100070953369, 0.6909594535827637, 0.42255833745002747, 0.6988
80498504639, 0.3660152554512024, 0.7042369246482849, 0.31870028376579285, 0.70371
059425354, 0.5349453091621399, 0.715964674949646, 0.46708035469055176, 0.72191619
7304688, 0.4208742380142212, 0.725350022315979, 0.3807961344718933], [0.638383746
471558, 0.6830650568008423, 0.6051630973815918, 0.6455150842666626, 0.59839636087
176, 0.5754604935646057, 0.640053391456604, 0.5366804599761963, 0.688016712665557
, 0.5331657528877258, 0.6132092475891113, 0.5027796030044556, 0.6246763467788696,
0.4250374436378479, 0.6350398659706116, 0.36972886323928833, 0.6457929015159607,
.32452017068862915, 0.6449153423309326, 0.4999355375766754, 0.6593978404998779, 0
41327670216560364, 0.6699190139770508, 0.35146790742874146, 0.6790083646774292, 0
29801657795906067, 0.6728963851928711, 0.5129991173744202, 0.6876707673072815, 0.
2989569902420044, 0.6962263584136963, 0.3718157112598419, 0.7038330435752869, 0.3
467395067214966, 0.6981992721557617, 0.5385384559631348, 0.7117862701416016, 0.47
71021795272827, 0.7187902331352234, 0.42568373680114746, 0.7235140204429626, 0.38
5402936935425]], 'labels': ['0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0'
 '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0
, '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
'0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
' , '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0', '0',
 '0', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '2', '1', '1', '1', '1',
1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1
, '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
'1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1',
', '1', '1', '1', '1', '1', '1', '1', '1', '1', '1']}
darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$
```

Data and Labels printed of data.pickle file:

The creation of a data.pickle file indicates that the script has serialized some data, the hand landmark coordinates information and save it to this file for future use. This serialized data can be loaded back into memory using the pickle.load()

Random Forest

It is like a team of decision-making trees. Each tree makes its prediction based on a random subset of data. Then, all the trees vote on the final prediction. This teamwork helps to reduce mistakes and make more accurate predictions.

Team of Trees: Random Forest combines many decision trees to make predictions.

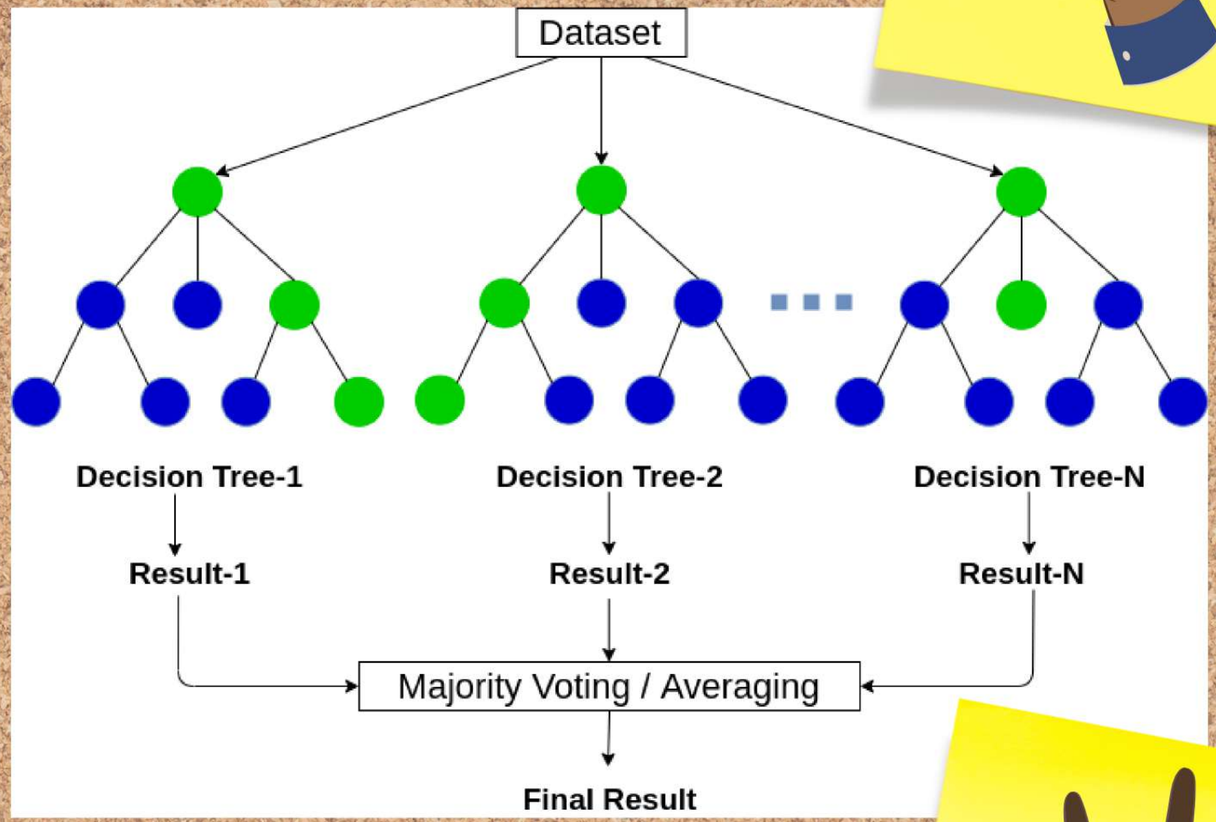
Randomness: Each tree only sees a random part of the data and considers a random set of features at each decision point.

Voting: All the trees vote on the final prediction, and the most popular choice wins.

Robustness: Because it's a team effort, Random Forest is less likely to make mistakes or overfit to the data.

Easy Tuning: You can adjust how many trees to include and how deep they grow to find the best balance between accuracy and efficiency.

Works for Different Problems: It can be used for both classification (like sorting objects into categories) and regression (like predicting prices).



Train Classifier (train_classifier.py):

Accuracy: 100%

- The classifier learned the training data perfectly without making any mistakes.
- It might have learned the data too well, including noise or unusual patterns.
- Test the classifier with new data to see if it still performs well.
- Make adjustments if needed to prevent overfitting(capture noise or random fluctuations).

Open | test_train_classifier.py ~/Desktop/Sign Language | Save

*temp.txt | test_create_dataset.py | test_detect_landmark.py | create_dataset.py | train_classifier.py | test_train_classifier.py

```python
import pickle
import numpy as np

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load data from pickle file
with open('./data.pickle', 'rb') as file:
    data_dict = pickle.load(file)

# Ensure consistent dimensions along dimension 1
data_arrays = data_dict['data']
data_lengths = [len(arr) for arr in data_arrays]

max_length = max(data_lengths)

# Pad arrays to the maximum length
data_padded = [np.pad(arr, (0, max_length - len(arr))) for arr in data_arrays]

# Convert data to NumPy array
data = np.vstack(data_padded)
labels = np.asarray(data_dict['labels'])

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, shuffle=True, stratify=labels)

# Initialize and train the Random Forest classifier
model = RandomForestClassifier()
model.fit(x_train, y_train)

# Make predictions on the test set
y_predict = model.predict(x_test)

# Evaluate the accuracy
score = accuracy_score(y_test, y_predict)
print('{}% of samples were classified correctly!'.format(score * 100))
```

darsh@ASUS-TUF-Dash-F15: ~/Desktop/Sign Language

darsh@ASUS-TUF-Dash-F15: ~/Desktop/Sign Language 103x8

```
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$ python3 test_train_classifier.py
100.0% of samples were classified correctly!
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$
```

Open | test_train_classifier.py ~/Desktop/Sign Language | Save

test_detect_landmark.py | create_dataset.py | train_classifier.py | test_train_classifier.py

```python
import pickle
import numpy as np

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

# Load data from pickle file
with open('./data.pickle', 'rb') as file:
    data_dict = pickle.load(file)

# Ensure consistent dimensions along dimension 1
data_arrays = data_dict['data']
data_lengths = [len(arr) for arr in data_arrays]

max_length = max(data_lengths)

# Pad arrays to the maximum length
data_padded = [np.pad(arr, (0, max_length - len(arr))) for arr in data_arrays]

# Convert data to NumPy array
data = np.vstack(data_padded)
labels = np.asarray(data_dict['labels'])

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, shuffle=True,
stratify=labels)

# Initialize and train the Random Forest classifier
model = RandomForestClassifier()
model.fit(x_train, y_train)

# Make predictions on the test set
y_predict = model.predict(x_test)

# Evaluate the accuracy
score = accuracy_score(y_test, y_predict)
print('{}% of samples were classified correctly!'.format(score * 100))

#Confusion Matrix: Examine the confusion matrix to understand which classes are being predicted accurately and
if there are any classes with lower performance.
conf_matrix = confusion_matrix(y_test, y_predict)
print("\nConfusion Matrix:")
print(conf_matrix)

#Feature Importance: If your data has many features, consider checking the feature importance provided by the
trained Random Forest model to understand which features contribute most to the predictions.
feature_importance = model.feature_importances_
print("\nFeature Importance:")
print(feature_importance)
```

Python 2 | Tab Width: 8 | Ln 47, Col 31 | INS

darsh@ASUS-TUF-Dash-F15: ~/Desktop/Sign Language

darsh@ASUS-TUF-Dash-F15: ~/Desktop/Sign Language 90x49

```
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$ python3 test_train_classifier.py
100.0% of samples were classified correctly!

Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0  2]]

Feature Importance:
[0.00129283 0.01734963 0.00021151 0.0110774  0.00509733 0.07584417
 0.02880308 0.08502761 0.03675835 0.06502895 0.00314379 0.11867306
 0.00049703 0.03975306 0.00200346 0.0152781  0.00388842 0.02099246
 0.00048014 0.02974625 0.         0.01396778 0.00403689 0.03116795
 0.01179115 0.01231077 0.00128228 0.06111708 0.00112131 0.0136912
 0.00681295 0.05917806 0.00706181 0.03889332 0.00081375 0.05512066
 0.00120846 0.0610391  0.         0.0185689  0.0023541  0.03751586
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.        ]
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$
```

Confusion Matrix:

- A confusion matrix is a table that summarizes the performance of a classification model.
- It shows the number of correct and incorrect predictions made by the model.
- Each row represents the actual class, while each column represents the predicted class.

Open | *train_classifier.py ~/Desktop/Sign Language | Save

temp.txt | create_dataset.py | *train_classifier.py

```python
import pickle
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

# Load data from pickle file
with open('./data.pickle', 'rb') as file:
    data_dict = pickle.load(file)

# Ensure consistent dimensions along dimension 1
data_arrays = data_dict['data']
data_lengths = [len(arr) for arr in data_arrays]

max_length = max(data_lengths)

# Pad arrays to the maximum length
data_padded = [np.pad(arr, (0, max_length - len(arr))) for arr in data_arrays]

# Convert data to NumPy array
data = np.vstack(data_padded)
labels = np.asarray(data_dict['labels'])

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, shuffle=True,
stratify=labels)

# Initialize and train the Random Forest classifier
model = RandomForestClassifier()
model.fit(x_train, y_train)

y_predict = model.predict(x_test) # Make predictions on the test set

score = accuracy_score(y_test, y_predict) # Evaluate the accuracy
print('{}% of samples were classified correctly!'.format(score * 100))

#Confusion Matrix: Examine to understand which classes are predicted accurately & which/if any with lower
performance.
conf_matrix = confusion_matrix(y_test, y_predict)
print("\nConfusion Matrix:")
print(conf_matrix)

#Feature Importance: If your data has many features, consider checking the feature importance provided by the
trained Random Forest model to understand which features contribute most to the predictions.
feature_importance = model.feature_importances_
print("\nFeature Importance:")
print(feature_importance)

f = open('model.p', 'wb')
pickle.dump({'model': model}, f)
f.close()
```

Python 2 | Tab Width: 8 | Ln 49, Col 10 | INS

darsh@ASUS-TUF-Dash-F15: ~/Desktop/Sign Language

darsh@ASUS-TUF-Dash-F15: ~/Desktop/Sign Language 90x49

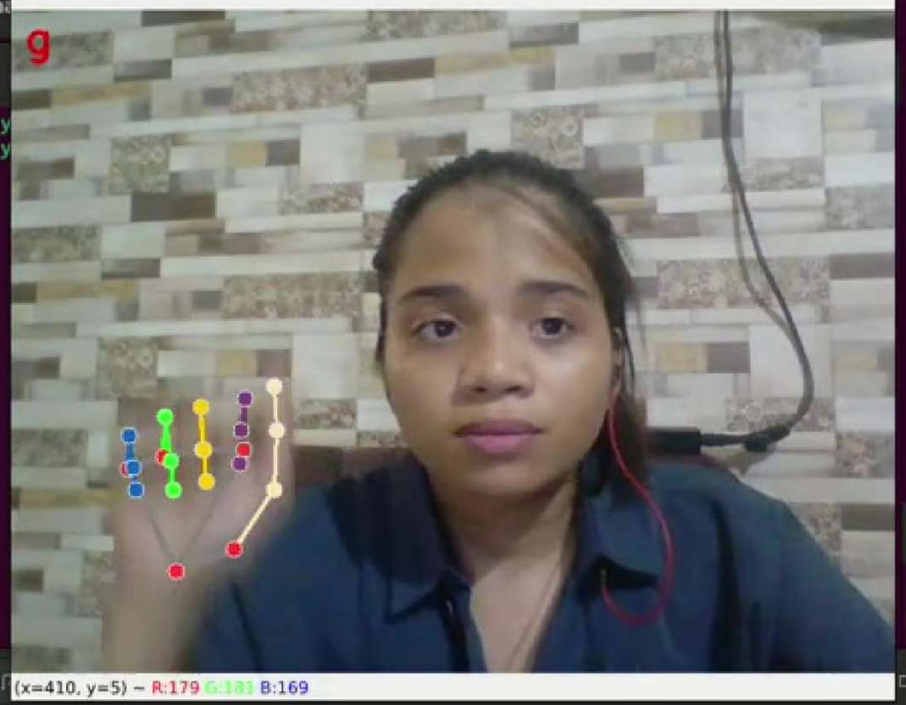
```
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$ python3 test_train_classifier.py
python3: can't open file '/home/darsh/Desktop/Sign Language/test_train_classifier.py': [Er
rno 2] No such file or directory
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$ python3 train_classifier.py
100.0% of samples were classified correctly!

Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0  2]]

Feature Importance:
[0.00027074 0.03202802 0.00049037 0.00250406 0.00019648 0.03851407
 0.03887376 0.09250961 0.05172045 0.034527   0.0007039  0.08394518
 0.00119663 0.02769547 0.00310441 0.01620681 0.00440945 0.02702281
 0.0007918  0.04057055 0.00147613 0.01373654 0.         0.0455633
 0.00830637 0.04370696 0.00368527 0.060083   0.00023563 0.0102023
 0.00270465 0.0344561  0.00199742 0.0475325  0.00214097 0.05854978
 0.00089412 0.103061   0.00028785 0.02492243 0.00745712 0.03171894
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.        ]
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$ ls
'ASL Signs.jpeg'     data.pickle   README.md                train_classifier.py
 collect_imgs.py     License       requirements.txt
 create_dataset.py   model.p       test_create_dataset.py
 data                myenv         test_detect_landmark.py
(myenv) darsh@ASUS-TUF-Dash-F15:~/Desktop/Sign Language$
```

"Classifier saved as model.p"

- It means that the trained machine learning model, which was created during the training process, has been saved or serialized into a file named "model.p".
- This allows you to reuse the trained model later without having to retrain it every time you want to make predictions on new data.
- Saving the model enables you to deploy it in other applications or environments where you need to use it for prediction tasks.

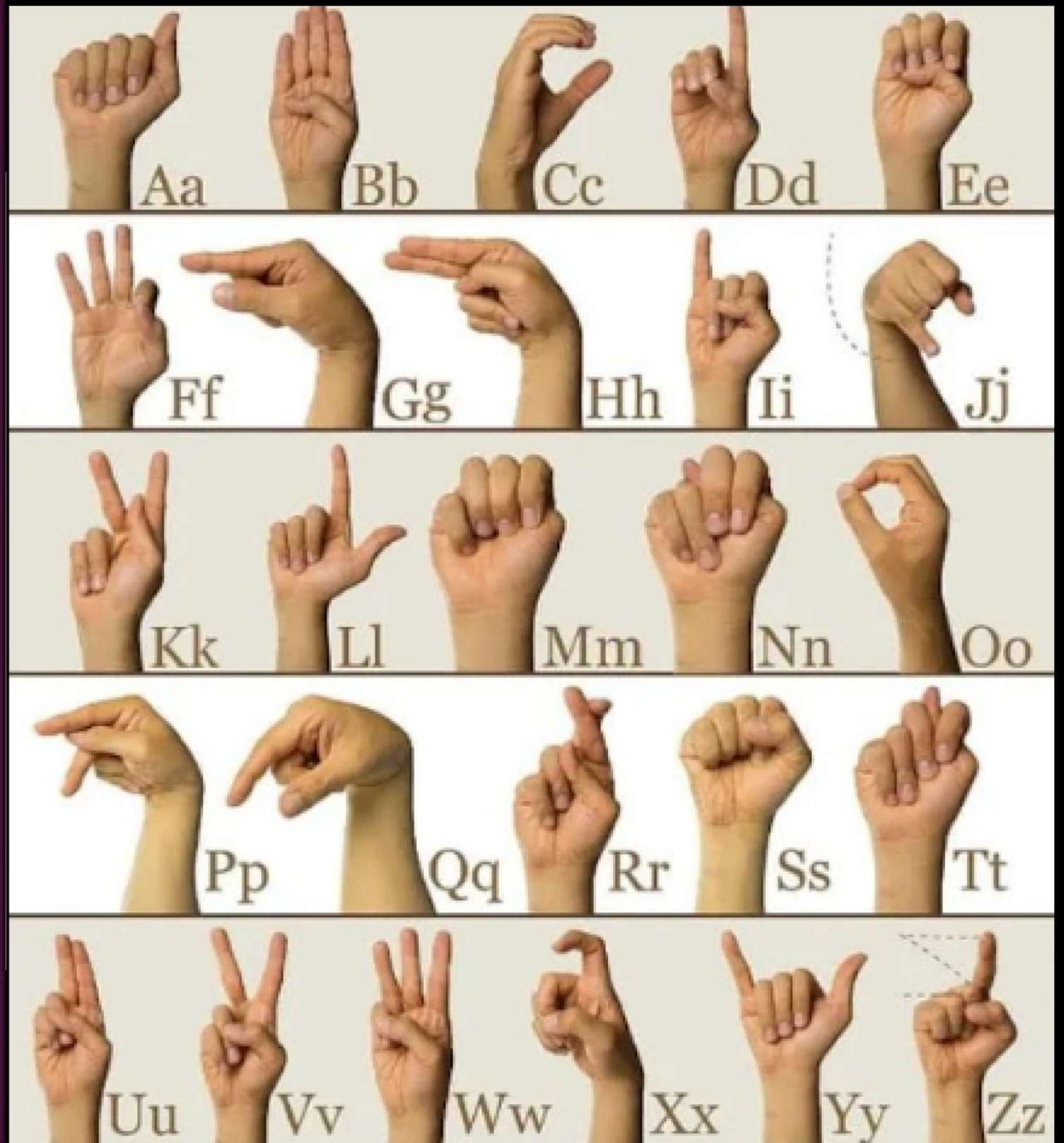
DATASET FROM A TO Z

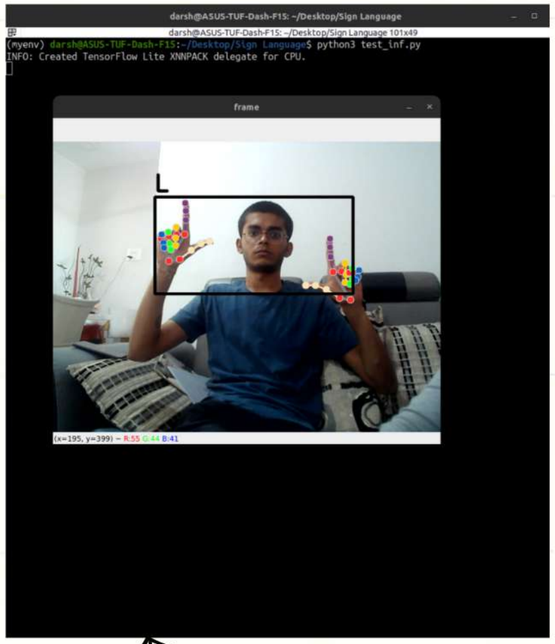
Frame

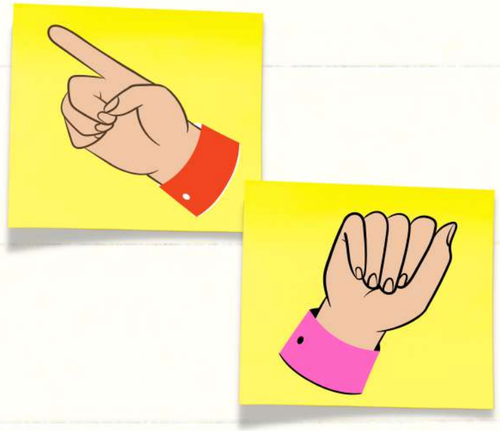


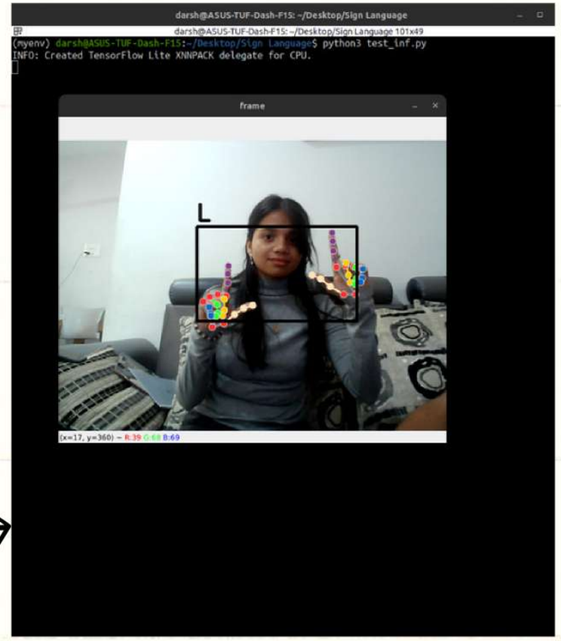

```
mance-e message_factory.GetMessageClass() instead. SymbolDatabase.GetPrototype() will
To enal be removed soon.
sorFlo  warnings.warn('SymbolDatabase.GetPrototype() is deprecated. Please '
2024-0:/home/aaryamourya/.local/lib/python3.10/site-packages/google/protobuf/symbol_dat
38] TF abase.py:55: UserWarning: SymbolDatabase.GetPrototype() is deprecated. Please us
WARNIN e message_factory.GetMessageClass() instead. SymbolDatabase.GetPrototype() will
STDERR be removed soon.
I0000 (  warnings.warn('SymbolDatabase.GetPrototype() is deprecated. Please '
ized E/home/aaryamourya/.local/lib/python3.10/site-packages/google/protobuf/symbol_dat
I0000 (abase.py:55: UserWarning: SymbolDatabase.GetPrototype() is deprecated. Please us
 ES 3..e message_factory.GetMessageClass() instead. SymbolDatabase.GetPrototype() will
INFO: (be removed soon.
/home/  warnings.warn('SymbolDatabase.GetPrototype() is deprecated. Please '
abase.:/home/aaryamourya/.local/lib/python3.10/site-packages/google/protobuf/symbol_dat
e mess abase.py:55: UserWarning: SymbolDatabase.GetPrototype() is deprecated. Please us
be rem e message_factory.GetMessageClass() instead. SymbolDatabase.GetPrototype() will
  warn be removed soon.
aaryam  warnings.warn('SymbolDatabase.GetPrototype() is deprecated. Please '
      /home/aaryamourya/.local/lib/python3.10/site-packages/google/protobuf/symbol_dat
      abase.py:55: UserWarning: SymbolDatabase.GetPrototype() is deprecated. Please us
      e message_factory.GetMessageClass() instead. SymbolDatabase.GetPrototype() will
      be removed soon.
        warnings.warn('SymbolDatabase.GetPrototype() is deprecated. Please '
```

signs.png









Open · test_inf.py · ~/Desktop/Sign Language · Save

temp.txt | collect_imgs.py | create_dataset.py | train_classifier.py | test_inf.py

```
    h, w, _ = frame.shape

    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    results = hands.process(frame_rgb)
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                frame,
                hand_landmarks,
                mp_hands.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style())

            # Extract features for normalization
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                x_.append(x)
                y_.append(y)

            # Extract normalized features
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                data_aux.append((x - min(x_)) * W)
                data_aux.append((y - min(y_)) * H)

            x1 = int(min(x_) * W) - 10
            y1 = int(min(y_) * H) - 10

            x2 = int(max(x_) * W) - 10
            y2 = int(max(y_) * H) - 10

            # Ensure the length of data_aux matches the expected number of features (84)
            if len(data_aux) == 84:
                prediction = model.predict([np.asarray(data_aux)])
                predicted_character = labels_dict[int(prediction[0])]

                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)
                cv2.putText(frame, predicted_character, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 0, 0), 3,
                            cv2.LINE_AA)

    cv2.imshow('frame', frame)
    cv2.waitKey(1)

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

Python 2 · Tab Width: 8 · Ln 74, Col 1 · INS

SUCCESSFUL DETECTION

Conclusion

Our project employs Python and OpenCV to create a real-time sign language recognition system using Machine Learning models. It accurately interprets American Sign Language gestures and converts them into text. Through rigorous training and evaluations, the system proves effective in aiding individuals with hearing impairments, fostering inclusivity and connectivity in society.

Thanks!