USCI

SAP

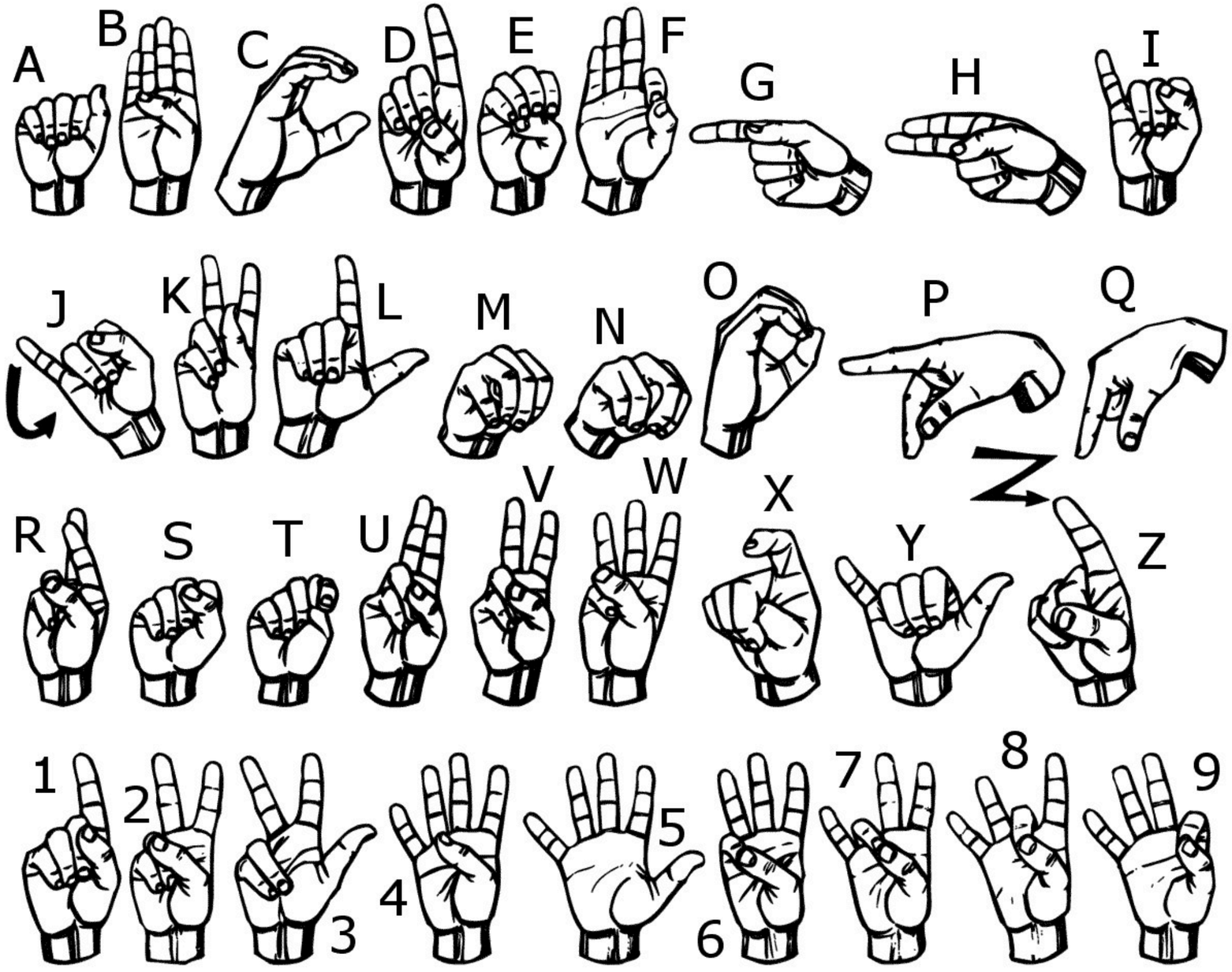
code unnati

Sign Language To Text

INTRODUCTION

A sign language is a language which uses gestures instead of sound to convey meaning combining hand-shapes, orientation and movement of the hands, arms or body, facial expressions and lip-patterns.

American sign language is a predominant sign language
Since the only disability Deaf and Dumb (hereby referred to as D&M) people have is communication related and **since they cannot use spoken languages, the only way for them to communicate is through sign language.**


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
1
2
3
4
5
6
7
8
9

APPROACH

- There are many approaches to solve the given problem.
- The **first approach involves building an image classifier** that takes the entire frame as input and labels it with the symbol to classify. Another approach involves cropping the image and considering only the symbol to detect.
- However, the **most promising approach** the is to **extract the hand's position information** through **landmark detection.**

we have used **landmark detection** with

Python
OpenCV
Mediapipe
& Scikit Learn

to classify sign language symbols.

METHODOLOGY

STEPS

Data Prepare

Data pre-processing

Data Training & Testing

Result

Data Preparation

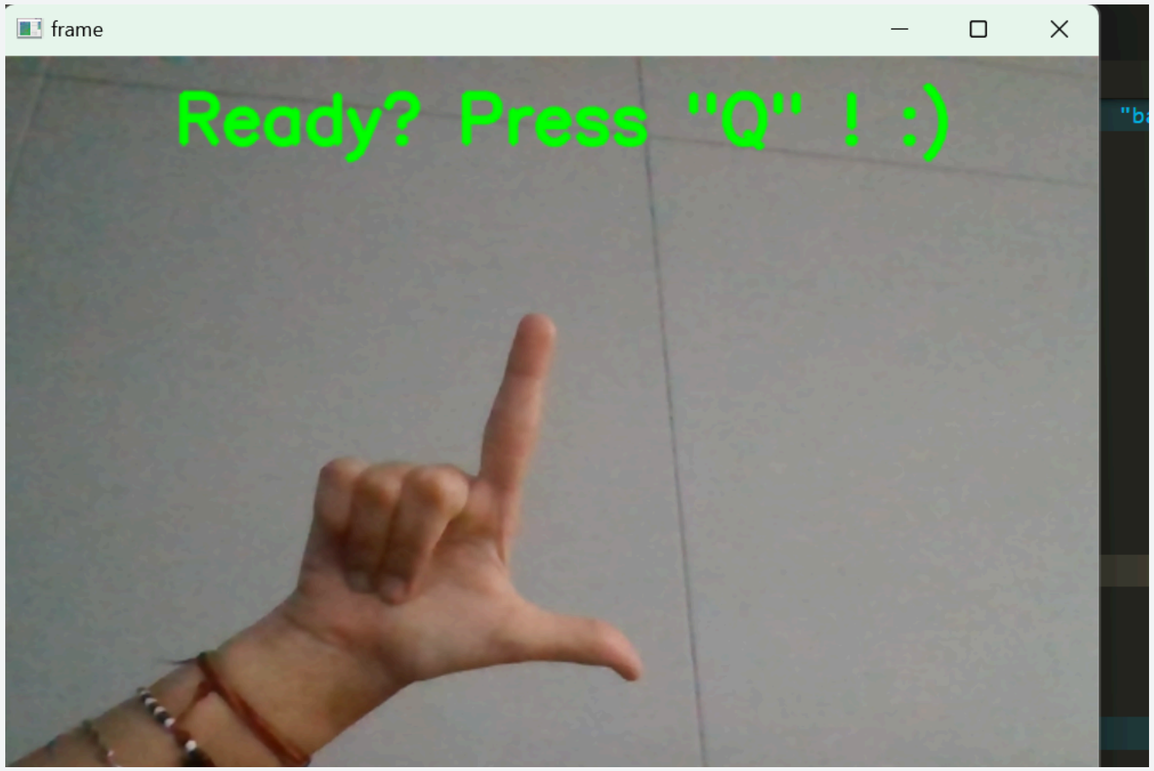
- In data collection process we captured a **diverse set** of **sign language gestures,** providing a foundation for **training a robust sign language detection** model. The dataset is structured in a way that facilitates effective model training, with each class containing a **sufficient number of images**.

- Here we will write Python script for collecting image data using a **webcam** for a sign language detection application.

- For webcam we will use **cv2**. cv2 is a module within the **OpenCV** library.

```python
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)

number_of_classes = 3
dataset_size = 100

cap = cv2.VideoCapture(0)
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))

    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)    "imshow": Unknown word.
        if cv2.waitKey(25) == ord('q'):
            break
```

This window will help to capture the images and store it into the file.




```
PS C:\Users\baruy> & C:/Users/baruy/AppData/Local/Programs/Python/
er/sign-language-detector-python-master/collect_imgs.py
Collecting data for class 0
Collecting data for class 1
Collecting data for class 2
PS C:\Users\baruy>
```

OUTPUT

Data Preprocessing

- In data pre-processing we process a dataset of hand images using the MediaPipe library to extract hand landmarks and save the processed data in a pickled file.

- **Mediapipe:**
  - MediaPipe is an open-source framework developed by Google designed to make it easier for developers to integrate machine learning models into their applications, especially those involving **computer vision** and multimedia processing.
  - **pickle** is a module in Python that provides a way to serialize and deserialize Python objects. It helps in enabling the storage or transmission of Python objects between different programs or across different platforms.

```
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:
        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)

        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_))
            data_aux.append(y - min(y_))

    data.append(data_aux)
    labels.append(dir_)
```

Necessary steps in preprocessing

- **cv2.imread**: This function from OpenCV is used to read an image from a file.
- **cv2.cvtColor:** Converts the color space of the image from BGR (Blue-Green-Red) to RGB (Red-Green-Blue).
- **hands.process:** Uses the mediapipe library to process the RGB image and detect hand landmarks.
- when hand landmarks are detected, it iterates over each set of hand landmarks and extracts the x and y coordinates of each landmark.

Data Training and Testing

- Pickle: Module for serializing and deserializing Python objects.
- In this step we will use **Random Forest Classifier.**
- Training a Random Forest Classifier using **scikit-learn** on a dataset to evaluate **its performance**, and saving the trained model.

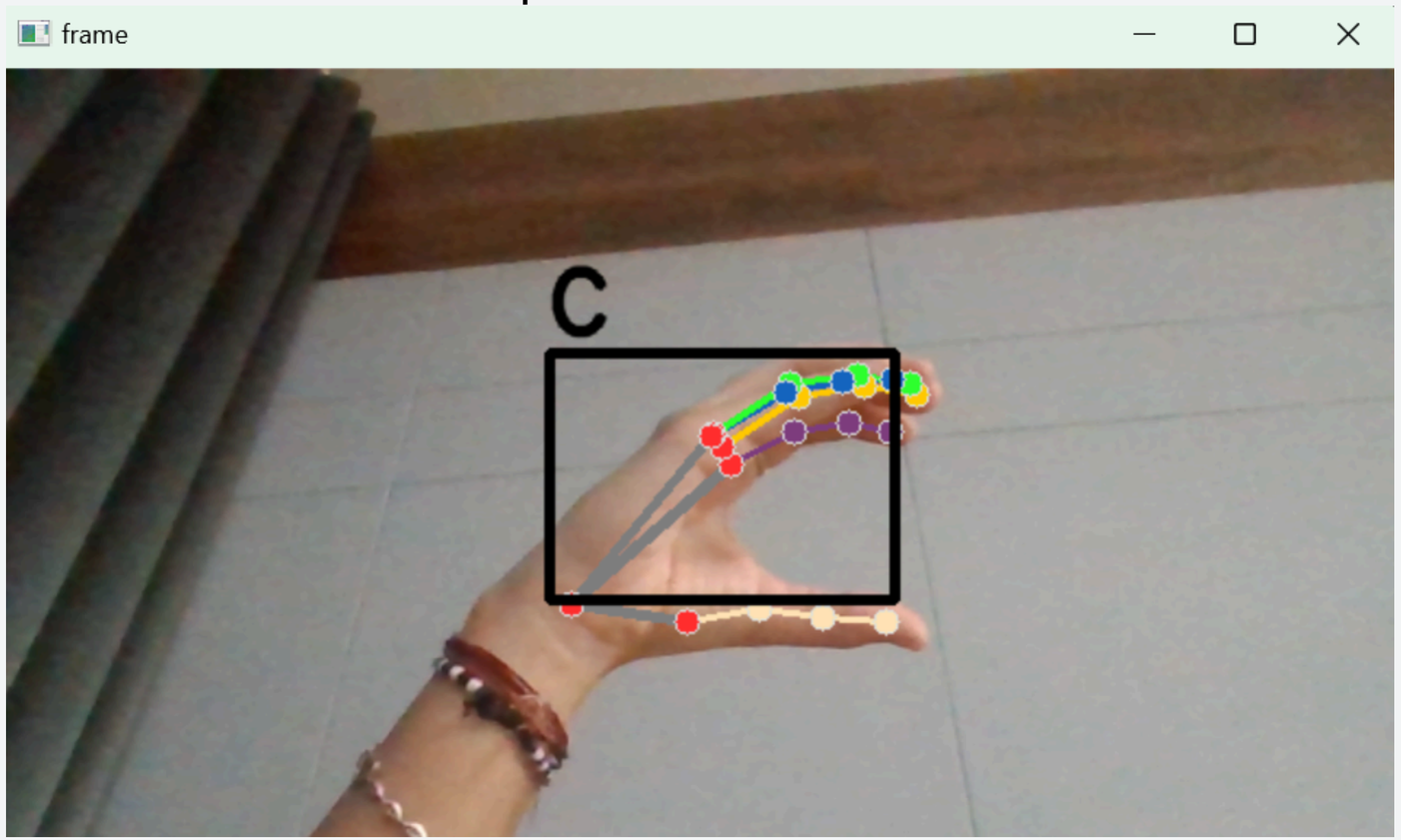
- For spliting the data we used **sklearn.model_selection** into testing and tranning.

- For training and testing a machine learning model. The data and labels are split into training and testing sets, and these sets are assigned to variables
  (x_train, x_test, y_train, and y_test)

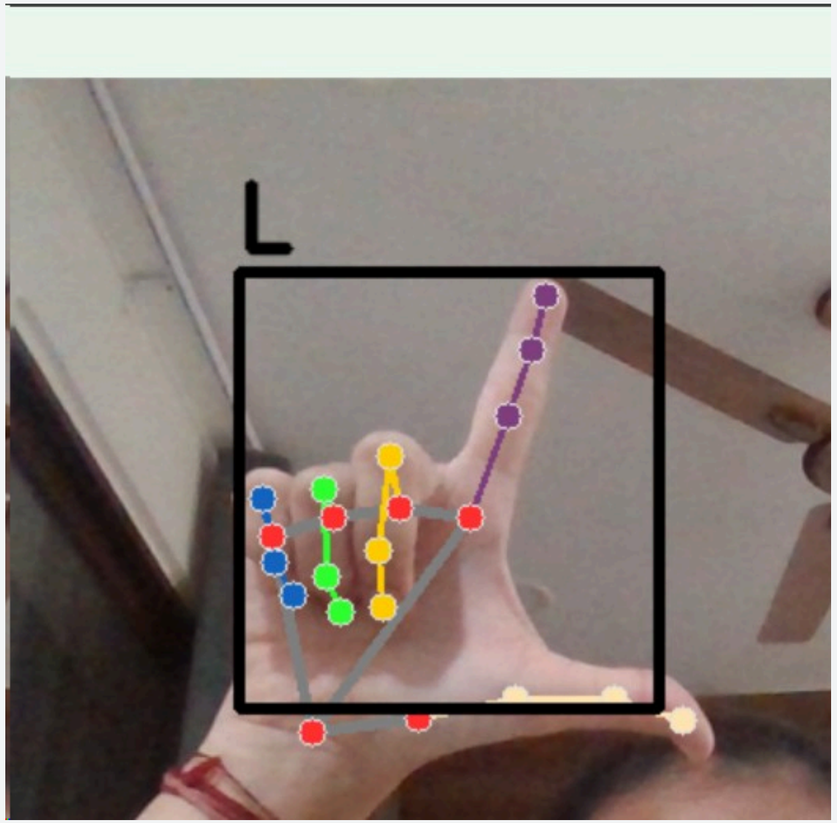
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np

#print(os.getcwd())
print(os.chdir('''C:\\Users\\baruy\\Downloads\\sign-language-detector-python-master\\sign-language-detector-python-master'''))


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

data = np.asarray(data_dict['data'])
labels = np.asarray(data_dict['labels'])

x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, shuffle=True, stratify=labels)

model = RandomForestClassifier()

model.fit(x_train, y_train)

y_predict = model.predict(x_test)

score = accuracy_score(y_predict, y_test)

print('{}% of samples were classified correctly !'.format(score * 100))

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

RESULT

The model successfully translated diverse sign language gestures into accurate text representations



During our evaluation, we successfully demonstrated the translation of sign language gestures into text, particularly showcasing the accurate recognition of the letter "A" and "L"formed using hand gestures. The model seamlessly interpreted the hand movements, and the corresponding "A"and "L" letter was displayed on the screen.



Future Goals

Looking forward, our Sign Language to Text Translation project has promising avenues for further development. Future goals include **refining the model's accuracy and expanding its vocabulary** to encompass a broader set of sign language gestures. Additionally, we aim to **enhance the system's real-time capabilities, making it more adaptable for dynamic communication scenarios.**

Ultimately, our goal is to create an inclusive and universally accessible communication tool that empowers individuals who use sign language in diverse settings.

Thank you!

Presented by:
Yashvi Baru
Maitri Patel
Anjali Sharma
Nidhi Jani