American Sign Language(ASL) Classification

•••

Group 5:
Yohannes Nigusse : Samantha Jerez-Roemer :
Amogh Hari Krishna: Robert Denning : Biniyam Fekede

Topic

AI-Powered Real-time Classification of Sign Language: This project aims to develop an AI model that recognizes sign language in real time. This innovation can significantly enhance the communication abilities of audibly impaired individuals. The topic is interesting because American Sign Language (ASL) is a complex and distinct language used by a significant portion of the population, particularly in the deaf community. By creating AI that can understand and communicate in sign language, we can slowly deconstruct the language barriers that marginalize the hard of hearing.

Essential Question

How can we use AI to help the hard of hearing communicate?

The number of Americans who are hard of hearing or deaf is disproportionate to the number of Americans who can understand American Sign Language(ASL). According to a recent study, there are approximately 2 million deaf people in the United States who use sign language as their primary mode of communication (www.icphs2019.org). Our group sees the potential of AI to better integrate those who are hard of hearing into the lives of those who don't have difficulty hearing.

Supporting Evidence

It is estimated that about 15% of the American population – or around 37.5 million people – are deaf or hard of hearing. This includes people who have a partial loss of hearing, as well as those who are completely deaf.

https://www.icphs2019.org/the-deaf-and-hard-of-hearing-population-in-america#:~:text=It%20is%20estimated%20that%20about%2015%25%20of%20the,as%20well%20as%20those%20who%20are%20completely%20deaf.

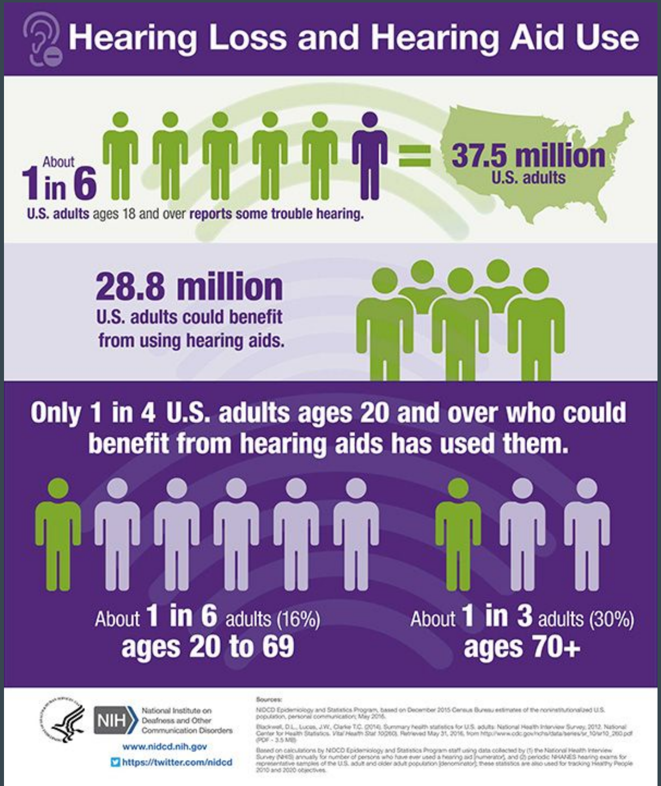
Research has shown that the exact number of American speakers of American Sign Language is hard to state, with estimates ranging between 500,000 and two million.

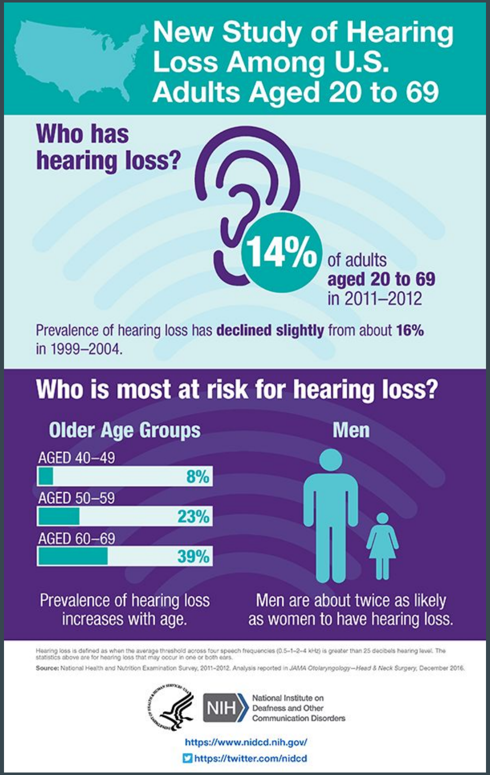
https://www.usahearingcenters.com/deaf-resources/sign-language-statistics.html

According to a recent study, there are approximately 2 million deaf people in the United States who use sign language as their primary mode of communication.

https://www.icphs2019.org/the-use-of-sign-language-among-deaf-people-in-the-united-states

Visuals (Tables, Charts, Graphics, Multimedia, etc)





Algorithm

The algorithm is split up into three main parts:
Gather data
Generate a data set
Train the model (in this case we used a random forest classifier) and test the model with real time data

Necessary libraries are:
Opencv a Python library that allows you to perform image processing and computer vision tasks

MediaPipe an open-source framework for building pipelines to perform computer vision inference over arbitrary sensory data such as video or audio

Scikit-learn a software machine learning library for Python

Gather Data

```
import os
import cv2

# Define the directory to store the collected data
DATA_DIR = './data'

# Create the directory if it does not exist
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)

# Define the number of classes (signs) and the dataset
size per class
number_of_classes = 26
dataset_size = 100

# Open a connection to the default camera (0) using
OpenCV, this may vary depending on the user
cap = cv2.VideoCapture(0)

# Loop through each class
for j in range(number_of_classes):
    # Create a directory for each class if it does not
exist
    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))
```

```
    # Display a message to prompt the user to press "Q" to
start collecting data
    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

    # Collect images for the current class
    counter = 0
    while counter < dataset_size:
        ret, frame = cap.read()
        cv2.imshow('frame', frame)
        cv2.waitKey(25)
        # Save the collected image to the corresponding class
directory
        cv2.imwrite(os.path.join(DATA_DIR, str(j),
'{}.jpg'.format(counter)), frame)

        counter += 1

# Release the camera connection and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()
```

Generate a data set

```
import os
import pickle
import mediapipe as mp
import cv2
import matplotlib.pyplot as plt
# Initialize MediaPipe Hands module
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=0.3)
# Define the directory where the collected images are stored
DATA_DIR = './data'
# Lists to store the extracted hand landmarks data and corresponding
labels
data = []
labels = []
# Loop through each subdirectory (class) in the data directory
for dir_ in os.listdir(DATA_DIR):
    # Loop through each image in the current subdirectory
    for img_path in os.listdir(os.path.join(DATA_DIR, dir_)):
        # List to store the hand landmarks data for the current image
        data_aux = []
        # Lists to store the x and y coordinates of hand landmarks
        x_ = []
        y_ = []
        # Read the image and convert it to RGB format
        img = cv2.imread(os.path.join(DATA_DIR, dir_, img_path))
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

```
        # Process the image to detect hand landmarks using MediaPipe
Hands
        results = hands.process(img rgb)
        # Check if hand landmarks are detected in the image
        if results.multi hand landmarks:
            # Loop through each detected hand in the image
            for hand landmarks in results.multi hand landmarks:
                # Extract x and y coordinates of each hand landmark
                for i in range(len(hand landmarks.landmark)):
                    x = hand landmarks.landmark[i].x
                    y = hand landmarks.landmark[i].y
                    # Store x and y coordinates in separate lists
                    x .append(x)
                    y .append(y)
                # Normalize the coordinates relative to the minimum
values
                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 ))
            # Append the hand landmarks data and corresponding label
to lists
            data.append(data aux)
            labels.append(dir )
# Save the extracted data and labels to a pickle file
f = open('data.pickle', 'wb')
pickle.dump({'data': data, 'labels': labels}, f)
f.close()
```

Train the model

```
import pickle

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

# Load the preprocessed data from the pickle file
data_dict = pickle.load(open('./data.pickle', 'rb'))

# Convert the data and labels to NumPy arrays
data = np.asarray(data_dict['data'])
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 a RandomForestClassifier
model
model = RandomForestClassifier()

# Train the model on the training data
model.fit(x_train, y_train)

# Predict labels for the test data
y_predict = model.predict(x_test)

# Calculate the accuracy of the model on
the test set
score = accuracy_score(y_predict, y_test)

# Print the accuracy of the model
print('{}% of samples were classified
correctly !'.format(score * 100))

# Save the trained model to a pickle file
f = open('model.p', 'wb')
pickle.dump({'model': model}, f)
f.close()
```

Use the model

```python
import pickle
import cv2
import mediapipe as mp
import numpy as np

# Load the trained model from the pickle file
model_dict = pickle.load(open('./model.p', 'rb'))
model = model_dict['model']

# Open a connection to the default camera (0) using OpenCV, again this may vary from
user to user
cap = cv2.VideoCapture(0)

# Initialize MediaPipe Hands module
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=0.3)

# Dictionary mapping numeric labels to corresponding hand signs (A-Z)
labels_dict = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F',
               6: 'G', 7: 'H', 8: 'I', 9: 'J', 10: 'K', 11: 'L',
               12: 'M', 13: 'N', 14: 'O', 15: 'P', 16: 'Q', 17: 'R',
               18: 'S', 19: 'T', 20: 'U', 21: 'V', 22: 'W', 23: 'X',
               24: 'Y', 25: 'Z'}

# Infinite loop to continuously capture frames from the camera
while True:
```

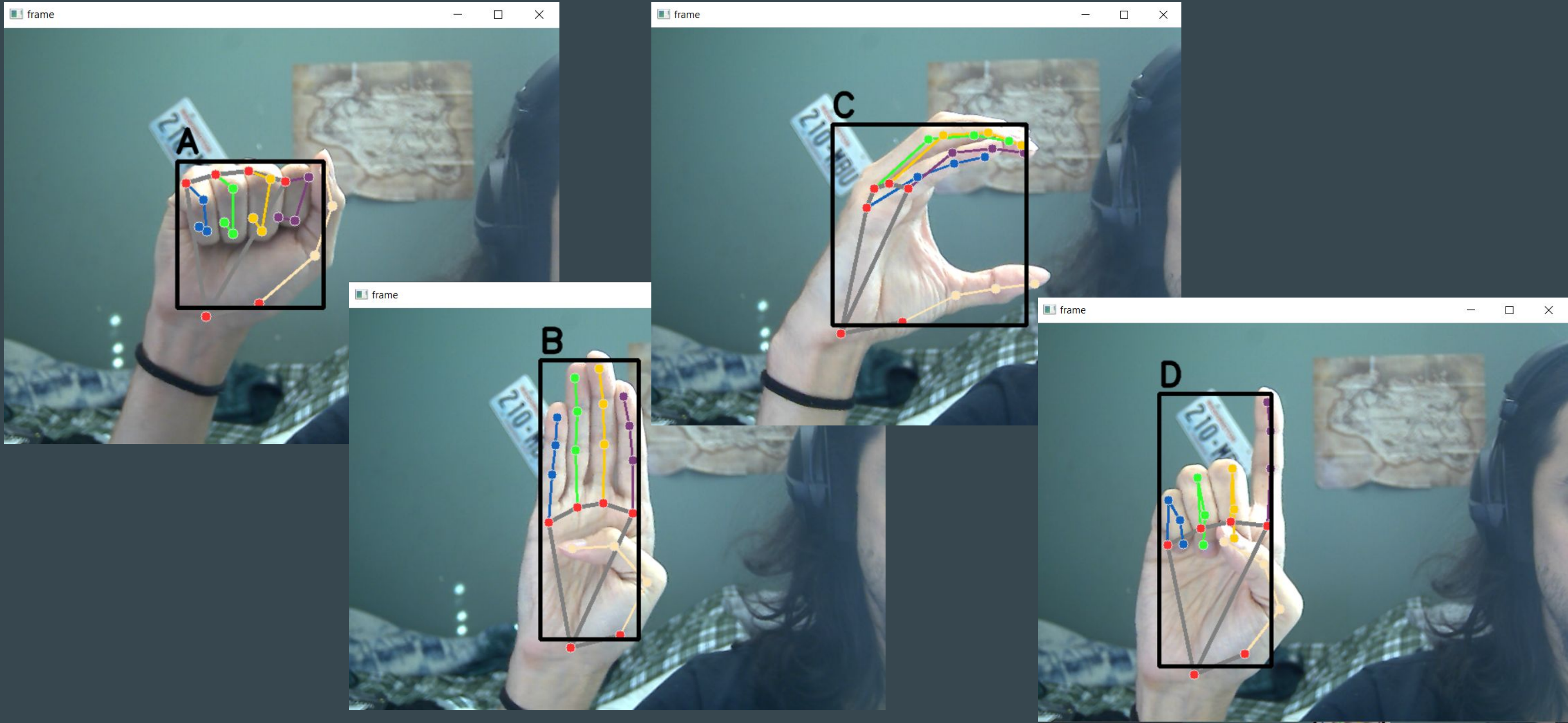
```python
    data_aux = []
    x_ = []
    y_ = []

    # Read a frame from the camera
    ret, frame = cap.read()

    # Get the height and width of the frame
    H, W, _ = frame.shape

    # Convert the frame to RGB format
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Process the frame to detect hand landmarks using MediaPipe
Hands
    results = hands.process(frame_rgb)
    # Check if hand landmarks are detected in the frame
    if results.multi_hand_landmarks:
        # Draw landmarks and hand connections on the frame
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                frame,  # image to draw
                hand_landmarks,  # model output
                mp_hands.HAND_CONNECTIONS,  # hand connections

mp_drawing_styles.get_default_hand_landmarks_style(),

mp_drawing_styles.get_default_hand_connections_style())
```

Use the model (cont.)

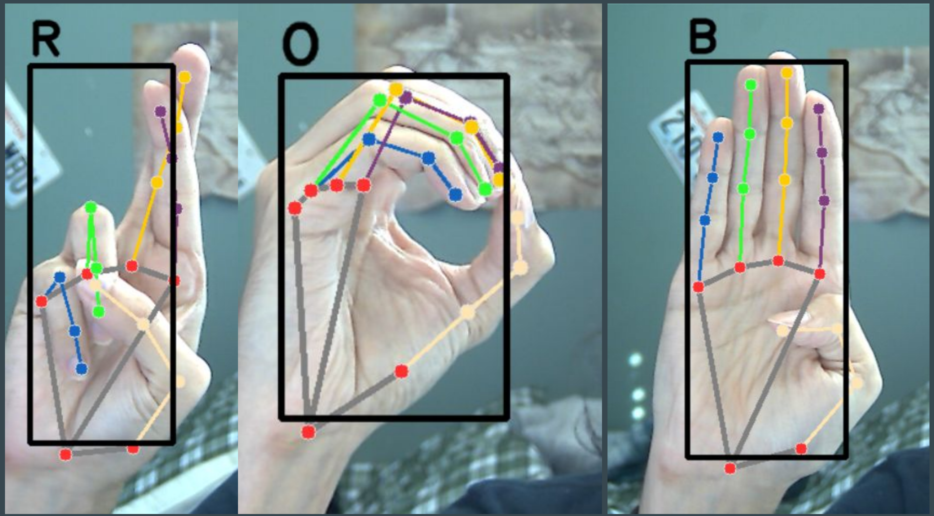
```
        # Extract hand landmarks data for further processing
        for hand_landmarks in results.multi_hand_landmarks:
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y

                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_))

        # Calculate bounding box coordinates for the hand region
        x1 = int(min(x_) * W) - 10
        y1 = int(min(y_) * H) - 10

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

```
        # Make a prediction using the trained model
            prediction = model.predict([np.asarray(data_aux)])
            # Get the predicted character corresponding to the numeric
label
            predicted_character = labels_dict[int(prediction[0])]
            # Draw a bounding box around the hand region and display
the predicted character
            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)
    # Display the frame with the drawn landmarks and bounding box
    cv2.imshow('frame', frame)
    # Check for the 'Esc' key to exit the infinite loop
    cv2.waitKey(1)

# Release the camera connection and close all OpenCV windows
cap.release()
cv2.destroyAllWindows()
```

Results



Results (cont.)



Next Steps for our Project

Increase diversity of the dataset. Currently the model is only trained on one member's hand. We need to account for different hand shapes/colors

ASL uses both hands and some of the hands signs have motions associated with their meaning. For example, J is similar to i, the difference being the motion of the letter J made with the pinky. The model needs to be improved for this

There are currently issues with, similar hand signs. I.e m, n, and t are similar and easily confused by the model, same with d and z

The UI could also be further improved to save letters/words to display full sentences

Citations

A. Nadaf and S. Pardeshi, "Classifying Sign Language Gestures using Decision Trees: A Comparison of sEMG and IMU Sensor Data," 2023 4th International Conference for Emerging Technology (INCET), Belgaum, India, 2023, pp. 1-8, doi: 10.1109/INCET57972.2023.10170736.

Aquino, Steven. “Inside Google’s Effort To Use AI To Make ASL Accessible To All.” *Forbes,* Forbes Magazine, 5 Oct. 2023, www.forbes.com/sites/stevenaquino/2023/05/30/inside-googles-effort-to-use-ai-to-make-asl-accessible-to-all/?sh=3ef71dd06bfa

Chavan, Amey & Deshmukh, Shubham & Favin Fernandes, “Sign Language Detection,” Arxiv org. https://arxiv.org/ftp/arxiv/papers/2209/2209.03578.pdf

Citations continued

Moujahid, Kemal El. “Machine Learning to Make Sign Language More Accessible.” *Google,* Google, 1 Dec. 2021, https://blog.google/outreach-initiatives/accessibility/ml-making-sign-language-more-accessible/

Papastratis, Ilias & Christos Chatzikonstantinou & Dimitrios Konstantinidis & Kosmas Dimitropoulos & Petros Daras. “Artificial Intelligence Technologies for Sign Language.” *Sensors (Basel, Switzerland),* U.S. National Library of Medicine, 30 Aug. 2021, www.ncbi.nlm.nih.gov/pmc/articles/PMC8434597/

Stenger, Marianne. “Learning Sign Language: 5 of the Most Powerful Benefits.” *Berlitz,* Berlitz, 19 Jan. 2022, www.berlitz.com/blog/benefits-learning-sign-language

Citations continued

Stoll, Stephanie. “How AI Could Help You Learn Sign Language.” *The Conversation*, 6 Oct. 2022, https://theconversation.com/how-ai-could-help-you-learn-sign-language-105708#:~:text=A%20CNN%20is%20a%20type,each%20point%20in%20the%20video.

Wen, Feng & Zixuan Zhang & Tianyiyi He & Chengkuo Lee. “AI Enabled Sign Language Recognition and VR Space Bidirectional Communication Using Triboelectric Smart Glove.” *Nature News,* Nature Publishing Group, 10 Sep. 2021, www.nature.com/articles/s41467-021-25637-w