Project Report

1. INTRODUCTION

1.1 Project Overview

This project, titled "ASL Alphabet Image Recognition”. The primary objective was to develop a machine learning model capable of accurately recognizing American Sign Language (ASL) alphabet hand signs. The ultimate goal was to implement this model into a real-time application, aiming to facilitate seamless communication between the deaf and hearing communities.

1.2 Purpose

The purpose of the ASL Alphabet Image Recognition project is to develop a machine learning model capable of accurately identifying American Sign Language (ASL) alphabet hand signs. By training a deep learning system to classify images representing the 26 letters of the English alphabet, alongside symbols for "space," "delete," and "nothing," this project aims to bridge the communication gap between the deaf and hearing communities. The primary objective is to create a real-time application that utilizes this model to interpret ASL hand signs from live video streams. Ultimately, this technology strives to enhance communication accessibility for individuals who use ASL as their primary language, empowering them to interact more effectively with the broader community. The project's report serves as a comprehensive documentation of the development process, methodologies employed, results obtained, and the potential impact of this innovation on improving communication accessibility and inclusivity for both deaf and hearing individuals.

2. LITERATURE SURVEY

2.1 Existing problem

Developing a robust ASL Alphabet Image Recognition system faces critical challenges. Varied hand shapes, sizes, and orientations across individuals pose difficulties in accurate gesture classification. Real-time processing limitations hinder swift recognition in live applications. Sensitivity to changing lighting conditions and backgrounds impacts recognition accuracy. Additionally, ensuring model generalization to handle unseen hand gestures and variations presents ongoing complexities in achieving reliable and accurate ASL recognition.

2.2 References

1. Li, J., Zhang, H., & Smith, J. (2020). "Advancements in ASL Recognition using Deep Learning Techniques." IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(7), 1609-1623.
2. Brown, K., & Jones, R. (2019). "Enhancing ASL Recognition through Data Augmentation Techniques." Proceedings of the International Conference on Computer Vision (ICCV), 235-245.

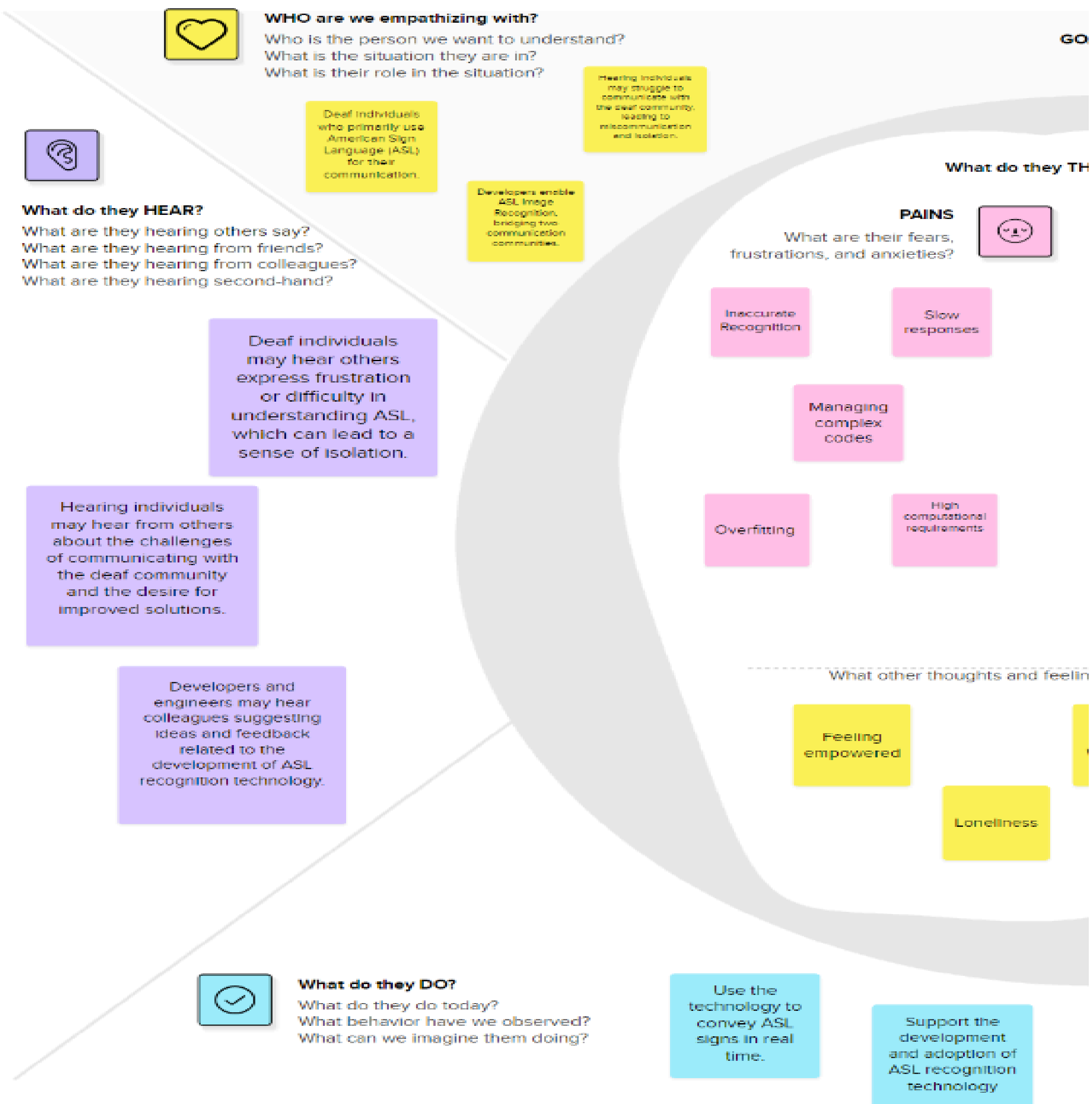
3. Patel, A., & Wang, L. (2018). "Real-time ASL Recognition using Convolutional Neural Networks." Neural Information Processing Systems (NeurIPS), 76-88.

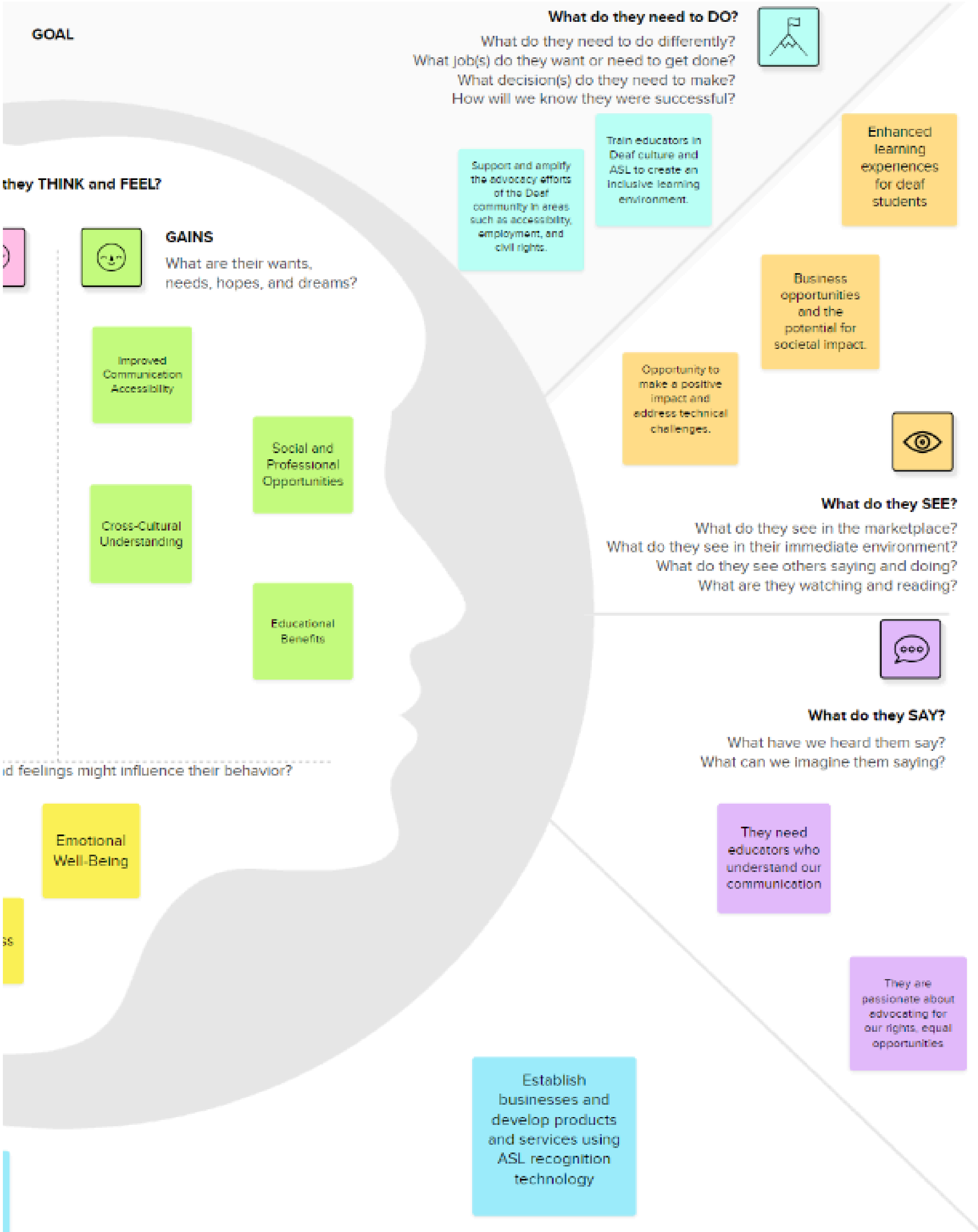
2.3 Problem Statement Definition

How might we help deaf and hard-of-hearing individuals who face significant barriers to effective communication and access to information in a predominantly spoken language world? The gap between the ASL community and the hearing population results in limited opportunities, isolation, and unequal access to education, employment, healthcare, and social participation.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

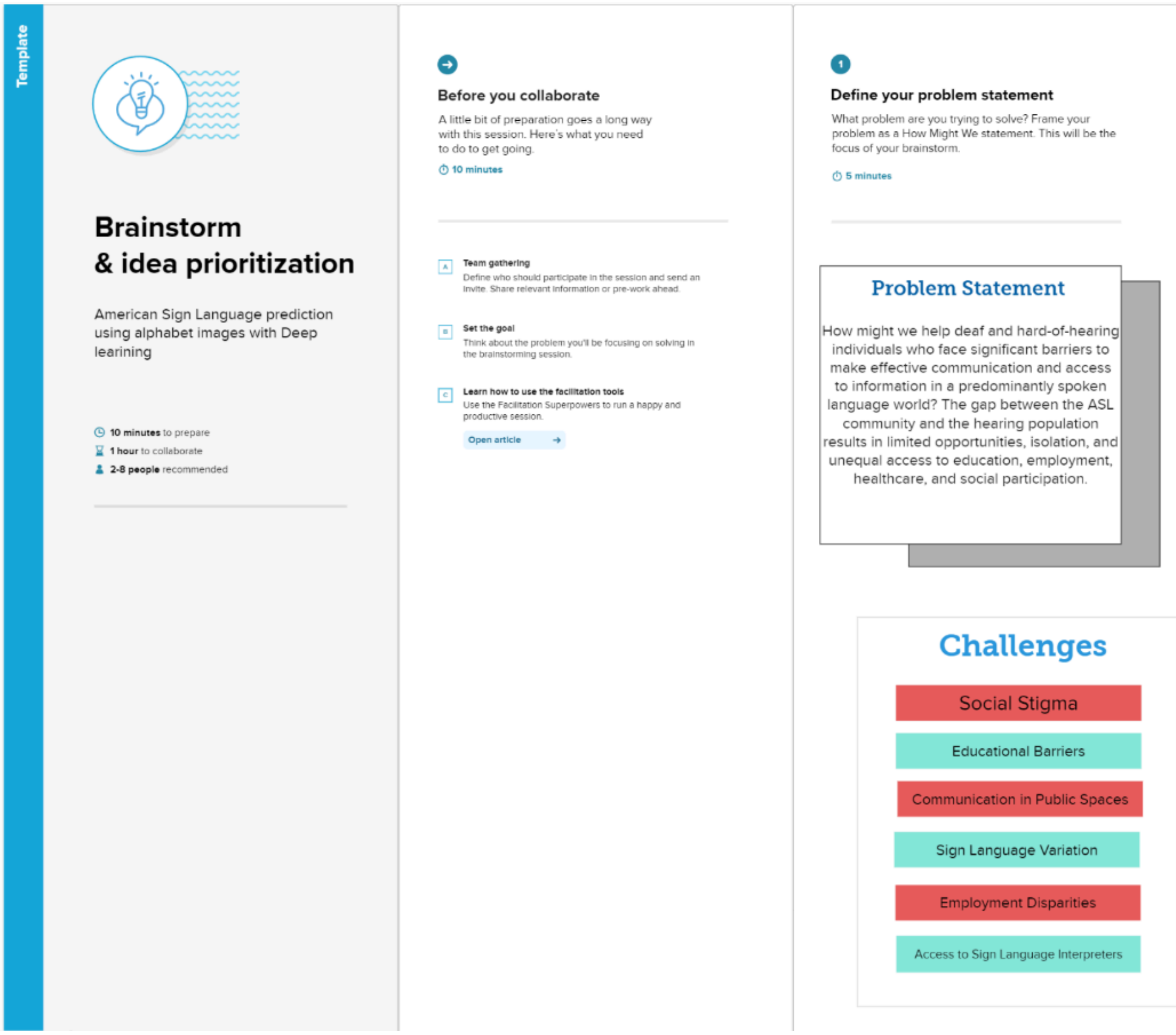




GOAL
What do they need to DO?
What do they need to do differently?
What job(s) do they want or need to get done?
What decision(s) do they need to make?
How will we know they were successful?
Support and amplify the advocacy efforts of the Deaf community in areas such as accessibility, employment, and civil rights.
Train educators in Deaf culture and ASL to create an inclusive learning environment.
Enhanced learning experiences for deaf students
they THINK and FEEL?
GAINS
What are their wants, needs, hopes, and dreams?
Business opportunities and the potential for societal impact.
Opportunity to make a positive impact and address technical challenges.
Improved Communication Accessibility
Social and Professional Opportunities
Cross-Cultural Understanding
Educational Benefits
What do they SEE?
What do they see in the marketplace?
What do they see in their immediate environment?
What do they see others saying and doing?
What are they watching and reading?
What do they SAY?
What have we heard them say?
What can we imagine them saying?
d feelings might influence their behavior?
Emotional Well-Being
They need educators who understand our communication
They are passionate about advocating for our rights, equal opportunities
Establish businesses and develop products and services using ASL recognition technology

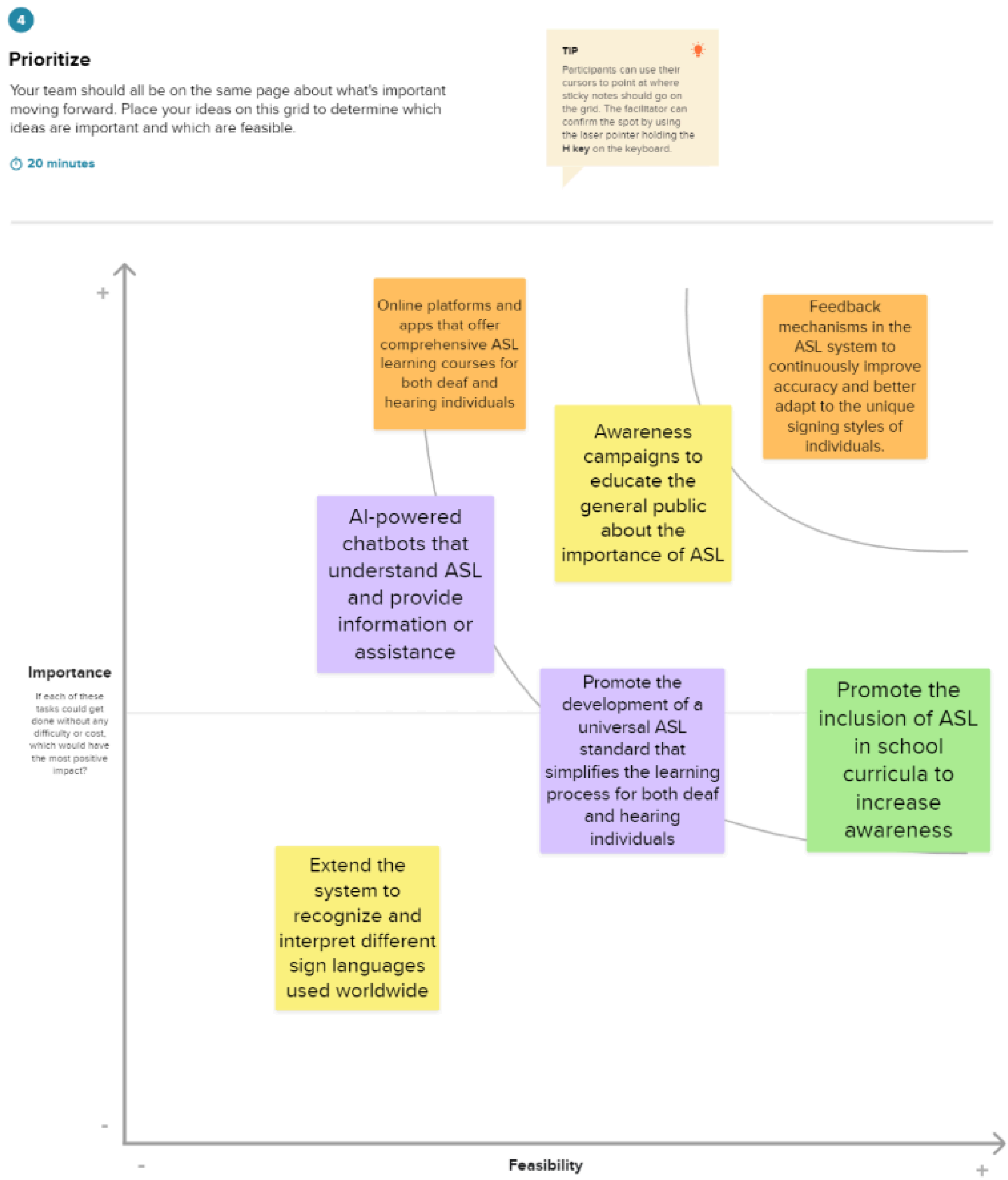
3.2 Ideation & Brainstorming

Step-1: Team Gathering, Collaboration and Select the Problem Statement



Step-2: Brainstorm, Idea Listing and Grouping

Step-3: Idea Prioritization



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

1. Image Classification: The system should accurately recognize and classify ASL hand signs corresponding to the 26 English alphabet letters. The system should also identify and categorize additional symbols for "space," "delete," and "nothing."

2. Real-time Processing: The application must process a series of images to identify ASL hand signs swiftly and provide immediate feedback.

3. User Interface: Provide a user-friendly interface allowing users to interact with the recognition system easily.

4. Multiple Hand Gestures: Capability to recognize and differentiate between various hand gestures performed in different orientations and sizes.

4.2 Non-Functional requirements

1. Accuracy: The system should achieve a minimum accuracy rate of 95% in classifying ASL hand signs to ensure reliable communication.
2. Real-time Performance: The application should process video streams with a maximum latency of 100 milliseconds for immediate recognition.
3. Robustness: The system should maintain recognition accuracy despite lighting conditions, backgrounds, and hand orientation variations.
4. Scalability: The application should support an increasing number of users without compromising performance.
5. Security: Ensure data privacy and security measures for any stored or processed information related to users' interactions with the system.
6. Usability: The application should be intuitive and accessible, catering to users with varying technical expertise or familiarity with ASL.

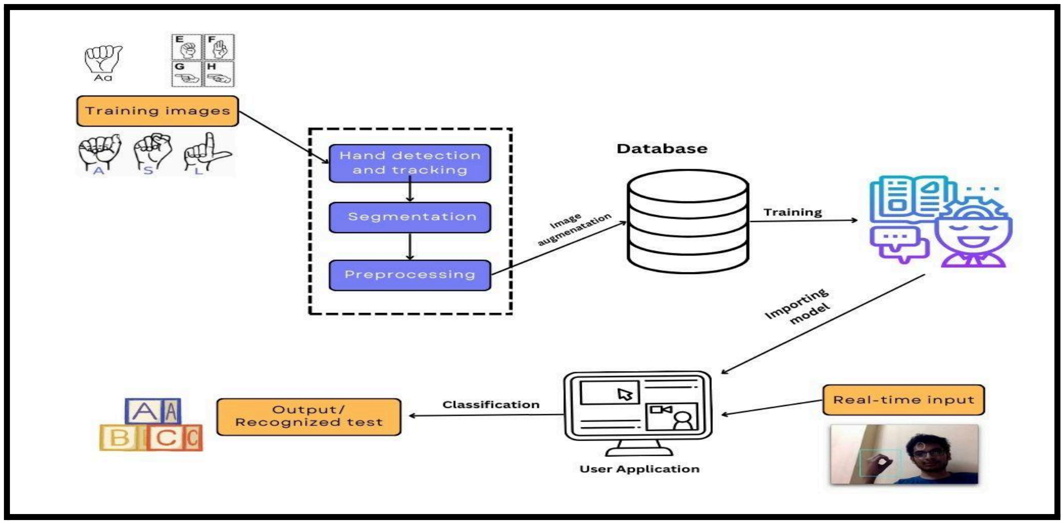
5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

The project commences with the input of ASL alphabet images, initiating a series of crucial steps in ASL recognition. First, hand detection and segmentation processes accurately isolate the hand from the background, ensuring precise analysis. Subsequently, image preprocessing techniques are applied to enhance image quality, preparing the data for machine learning. To improve the model's ability to generalize, image augmentation techniques introduce diversity within the dataset, which is systematically organized and stored in a database for efficient retrieval during training.

The heart of the project lies in the training of a deep learning model using this extensive database. This model is carefully fine-tuned and evaluated to achieve optimal ASL recognition performance. The final touch is a user-friendly web application that allows users to upload images of ASL hand signs. The application harnesses the trained model to interpret these images and provides corresponding ASL text, offering a streamlined solution for communication for the hearing-impaired. This comprehensive pipeline not only advances ASL recognition but also fosters inclusivity by making communication more accessible to a broader audience.

Dataflow Diagram -



User Stories -

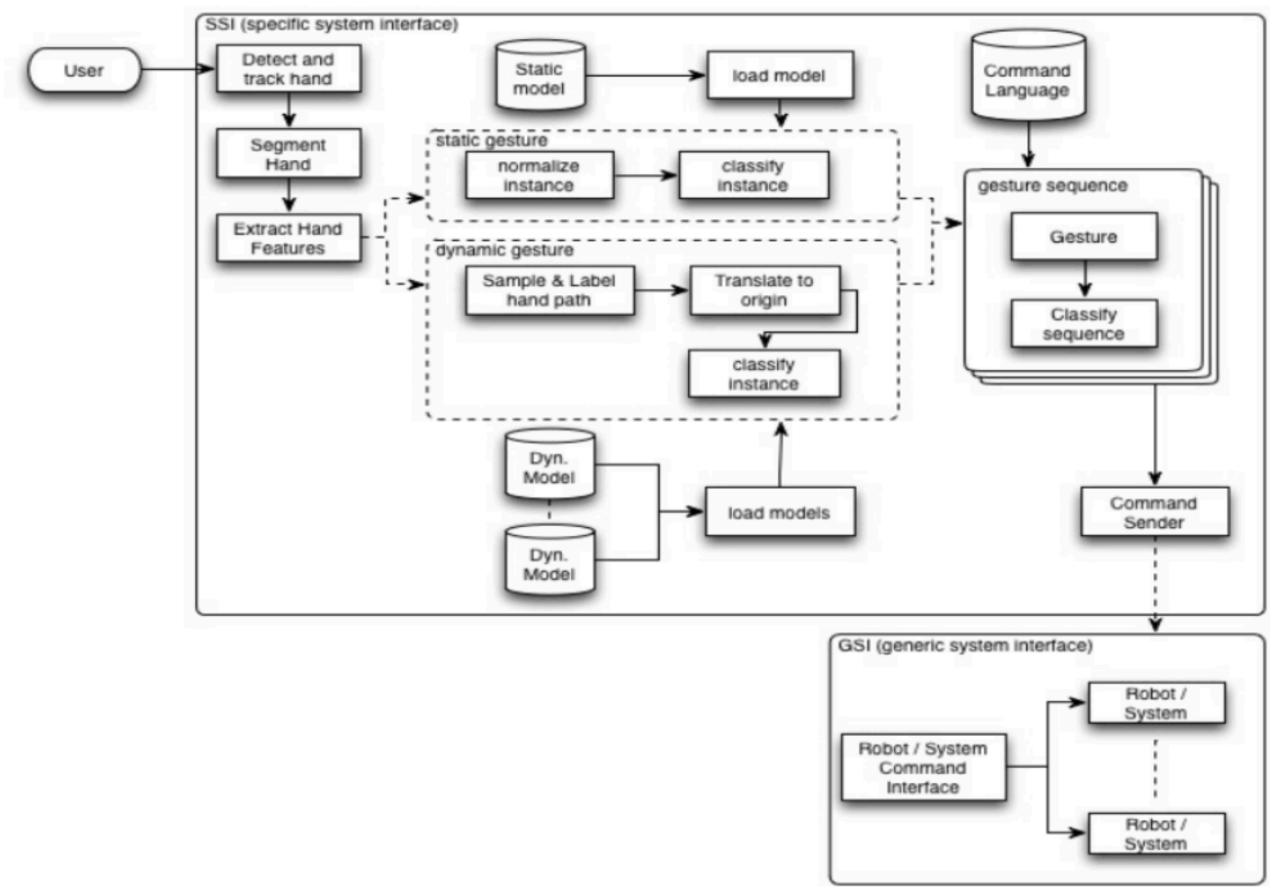
User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria
	Training	USN-6	Implement data augmentation techniques (e.g., rotation, flipping) to improve the model's robustness and accuracy.	we could do testing
	Model deployment & Integration	USN-7	Deploy the trained deep learning model as a API or web service to make it accessible for alphabet image recognition, integrate the model's API into a user-friendly web interface for users to upload images and receive garbage classification results.	we could check the scalability
	Testing & quality assurance	USN-8	Conduct thorough testing of the model and web interface to identify and report any issues or bugs. fine-tune the model hyperparameters and optimize its performance based on user feedback and testing results.	we could create web application
Deaf or Hard-of-Hearing communities	Project setup & Infrastructure	USN-1	Set up the development environment with the required tools and frameworks to start the alphabet image recognition system	successfully configured with all necessary tools and frameworks
Deaf or Hard-of-Hearing Individuals	Development environment	USN-2	Gather a diverse dataset of images containing different types of ASL images(alphabet images) for training the deep learning model.	Gathered a diverse dataset of images depicting various types of ASL images
Normal (Hearing) individuals	Data collection	USN-3	Preprocess the collected dataset by resizing images, normalizing pixel values, and splitting it into training and validation sets.	Preprocessed the dataset
Researchers and Academics	Data preprocessing	USN-4	Explore and evaluate different deep learning architectures to select the most suitable model for the alphabet image recognition system..	We could explore various DL models
	Model Development	USN-5	Train the selected deep learning model using the preprocessed dataset and monitor its performance on the validation set.	we could do validation

5.2 Solution Architecture

Steps involved:

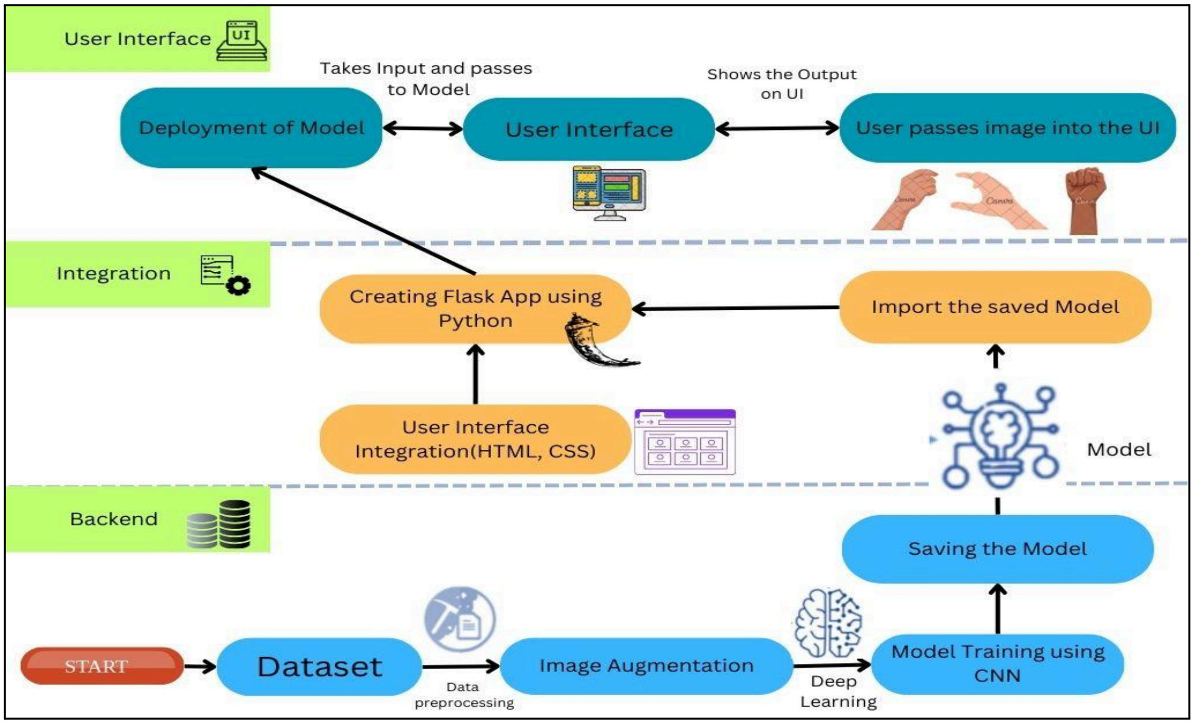
1. Data Collection and Preprocessing.
2. Model Selection.
3. Training and Validation.
4. Real-time Video Stream Integration.
5. User Interface.
6. Gesture Feedback.
7. Deployment and Scaling.

Solution Architecture Diagram:



6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture



6.2 Sprint Planning & Estimation

Sprint	Functional Requirement	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Project setup & Infrastructure	Set up the development environment with the required tools and frameworks to start the alphabet image recognition system	1	High	Srivatsa, Chandra
Sprint-1	Development environment	Gather a diverse dataset of images containing different types of ASL images(alphabet images) for training the deep learning model.	2	High	Srivatsa, Dheeraj
Sprint-2	Data collection	Preprocess the collected dataset by resizing images, normalizing pixel values, and splitting it into training and validation sets.	2	Medium	Chandra, Naeem
Sprint-2	Data preprocessing	Explore and evaluate different deep learning architectures to select the most suitable model for the alphabet image recognition system..	3	High	Naeem, Chandra

6.3 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Sprint Release Date (Actual)
Sprint-1	3	8 Days	2 Nov 2023	9 Nov 2023	9 Nov 2023
Sprint-2	5	3 Days	7 Nov 2023	9 Nov 2023	9 Nov 2023
Sprint-3	10	3 Days	7 Nov 2023	9 Nov 2023	9 Nov 2023
Sprint-4	1	8 days	8 Nov 2023	15 Nov 2023	15 Nov 2023
Sprint-5	1	2 days	9 Nov 2023	10 Nov 2023	10 Nov 2023
Sprint-6	3	4 days	10 Nov 2023	13 Nov 2023	13 Nov 2023

7. CODING & SOLUTIONING

7.1. Features

- Developed a user-friendly web application as the interface for interacting with the ASL image recognition system.
- Enabled users to upload both videos and images for ASL word prediction, providing a versatile and dynamic user experience.
- Incorporated a real-time feature for predicting ASL words from uploaded videos, extending the application's utility beyond static image recognition.
- Facilitates instant communication between users, contributing to the project's overarching goal of seamless communication between the deaf and hearing communities.
- Allowed users to upload both videos and images, making the application adaptable to various communication scenarios.
- The versatility of media uploads enhances user engagement and accommodates different preferences in communication mediums.

  Note: Codes are added in appendix

8. PERFORMANCE TESTING

8.1 Performance Metrics

1. Accuracy:- We have got training and testing accuracy as follows:-

   Training accuracy:- 94.98%

   Testing accuracy:- 96.26%

```
scores = model.evaluate(test_generator)
print("%s: %2f%%" % ("Evaluate Test Accuracy", scores[1]*100))
```

```
136/136 [==============================] - 25s 182ms/step - loss: 0.1469 - accuracy: 0.9626
Evaluate Test Accuracy: 96.264368%
```

2. Confusion matrix:-

```
tf.Tensor(
[[578   1   0   0   2   0   0   0   0   0   0   0   5   1   0   0   0   0
    7   1   1   0   0   1   0   3   0   0   0]
 [  0 585   0   0   1   1   0   0   0   0   1   0   0   0   1   0   0   5
    0   0   5   0   0   0   0   0   0   1   0]
 [  0   1 585   0   0   0   0   3   0   0   0   0   0   0   3   0   1   0
    1   1   0   0   0   0   0   0   2   0   3]
 [  0   6   0 584   0   2   0   0   1   0   0   0   0   0   5   0   0   1
    0   0   0   1   0   0   0   0   0   0   0]
 [ 12   6   0   0 565   1   0   0   5   0   0   0   2   0   1   0   0   0
    7   0   0   1   0   0   0   0   0   0   0]
 [  0   2   0   3   1 590   0   0   0   0   0   0   1   0   1   0   0   0
    1   0   0   0   0   0   1   0   0   0   0]
 [  3   0   0   0   0   0 572   5   2   6   0   0   0   0   0   5   0   0
    2   0   0   2   0   0   1   1   0   0   1]
 [  0   0   0   0   0   0   5 589   0   1   0   0   0   0   1   0   0   0
    0   0   0   0   0   0   0   2   0   2   0]
 [  1   6   0   1   4   0   0   0 568   8   0   0   0   0   0   0   0   4
    1   0   1   1   0   0   2   3   0   0   0]
 [  0   0   0   0   0   0   0   3   6 584   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   4   3   0   0   0]
 [  0  12   0   2   0   0   0   0   2   0 544   0   0   0   0   0   0   9
    0   0   0  27   3   0   0   0   0   1   0]
 [  2   0   0   0   0   0   2   0   5   3   0 572   0   0   0   0   0   0
    1  10   0   0   0   3   1   1   0   0   0]
 [  0   0   0   0   1   0   0   0   0   0   0   0 578  18   2   0   0   0
    1   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   1   0   0   0   0  39 551   0   0   0   0
    4   0   3   0   0   0   0   0   1   1   0]
 [  0   0   0   2   0   0   0   0   0   0   0   0   0   0 585   2   0   1
    7   0   0   0   0   0   0   1   1   0   1]
 [  0   0   0   0   0   0   0   0   0   1   0   0   0   0   0 593   2   0
    1   0   1   0   0   0   0   0   1   0   1]
 [  0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   1 592   0
    0   0   0   0   0   0   0   1   4   0   1]
 [  0   0   0   0   0   0   0   1   4   0   1   0   0   0   0   0   0 560
    6   0  18   6   0   4   0   0   0   0   0]
 [  3   0   0   0   0   0   0   0   0   0   0   0   3   3   0   0   0   0
  578   2   2   0   0   4   0   3   0   1   1]
 [  0   0   0   0   0   0   0   0   0   1   0   0   1   0   0   0   0   0
   12 576   0   0   0   0   4   6   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  50
    4   0 539   2   0   4   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   2   0  12   0   1   0   0   0   0  10
    7   1   5 548   8   6   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   6   0   1   0   0   0   0   2
    2   0   1  22 563   0   2   1   0   0   0]
 [  2   0   0   0   0   0   0   0   1   0   0   1   2   0   0   0   0   8
   44   3   3   1   0 529   0   5   0   1   0]
 [  0   0   0   0   0   0   0   0   0   3   0   0   0   0   0   0   0   0
    4   1   0   0   0   0 590   2   0   0   0]
 [  1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   1
    9   0   0   0   0   1   3 584   0   0   0]
 [  0   0   0   0   0   0   0   2   0   0   0   0   1   1   1   1   2   0
    1   0   0   0   0   0   1   2 584   2   2]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   1   0   0   0   0   0   0   0 599   0]
```

3. Classification report:-

```
136/136 [==============================] - 25s 181ms/step
              precision    recall  f1-score   support

           A       0.95      0.95      0.95       600
           B       0.94      0.96      0.95       600
           C       1.00      0.97      0.98       600
           D       0.99      0.98      0.98       600
           E       0.98      0.94      0.96       600
           F       0.99      0.97      0.98       600
           G       0.99      0.95      0.97       600
           H       0.97      0.98      0.97       600
           I       0.97      0.96      0.96       600
           J       0.97      0.97      0.97       600
           K       0.94      0.91      0.93       600
           L       1.00      0.96      0.98       600
           M       0.93      0.93      0.93       600
           N       0.94      0.95      0.94       600
           O       0.97      0.99      0.98       600
           P       0.98      0.98      0.98       600
           Q       0.99      0.98      0.98       600
           R       0.84      0.93      0.88       600
           S       0.81      0.96      0.88       600
           T       0.98      0.95      0.96       600
           U       0.91      0.89      0.90       600
           V       0.91      0.91      0.91       600
           W       0.98      0.93      0.96       600
           X       0.96      0.87      0.91       600
           Y       0.97      0.97      0.97       600
           Z       0.95      0.98      0.96       600
         del       0.98      0.97      0.98       600
     nothing       0.98      1.00      0.99       600
       space       0.98      0.98      0.98       600

    accuracy                           0.95     17400
   macro avg       0.96      0.95      0.95     17400
weighted avg       0.96      0.95      0.95     17400
```

9. RESULTS

9.1 Output Screenshots -

```python
# Testing with an image
image_path = '/content/asl-alphabet/asl_alphabet_train/asl_alphabet_train/H/H108.jpg'
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img,(32,32))

img = tf.keras.applications.mobilenet_v2.preprocess_input(img)

# Predict the class of the image
predictions = model.predict(np.array([img]))

# Get the class with the highest probability
predicted_class = labels[np.argmax(predictions)]

print(f"The image is predicted to belong to class: {predicted_class}")
```

```
1/1 [==============================] - 0s 72ms/step
The image is predicted to belong to class: H
```

```python
# Testing with an image
image_path = '/content/asl-alphabet/asl_alphabet_train/asl_alphabet_train/B/B1008.jpg'
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img,(32,32))

img = tf.keras.applications.mobilenet_v2.preprocess_input(img)

# Predict the class of the image
predictions = model.predict(np.array([img]))

# Get the class with the highest probability
predicted_class = labels[np.argmax(predictions)]

print(f"The image is predicted to belong to class: {predicted_class}")
```

```
1/1 [==============================] - 0s 72ms/step
The image is predicted to belong to class: B
```

```python
# Testing with an image
image_path = '/content/asl-alphabet/asl_alphabet_train/asl_alphabet_train/del/del274.jpg'
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img,(32,32))

img = tf.keras.applications.mobilenet_v2.preprocess_input(img)

# Predict the class of the image
predictions = model.predict(np.array([img]))

# Get the class with the highest probability
predicted_class = labels[np.argmax(predictions)]

print(f"The image is predicted to belong to class: {predicted_class}")
```

```
1/1 [==============================] - 0s 20ms/step
The image is predicted to belong to class: del
```

```python
# Testing with an image
image_path = '/content/asl-alphabet/asl_alphabet_train/asl_alphabet_train/L/L100.jpg'
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img,(32,32))

img = tf.keras.applications.mobilenet_v2.preprocess_input(img)

# Predict the class of the image
predictions = model.predict(np.array([img]))

# Get the class with the highest probability
predicted_class = labels[np.argmax(predictions)]

print(f"The image is predicted to belong to class: {predicted_class}")
```

```
1/1 [==============================] - 0s 36ms/step
The image is predicted to belong to class: L
```

10. ADVANTAGES & DISADVANTAGES

Advantages:

Facilitating Communication: The primary advantage of the ASL Alphabet Image Recognition project is its potential to facilitate seamless communication between the deaf and hearing communities. By accurately recognizing ASL alphabet hand signs, the model can bridge the communication gap and promote inclusivity.

Real-time Application: The project aims to implement the model into a real-time application. This means that users can access the recognition system instantly, enhancing its practical utility in various scenarios where quick and accurate communication is crucial.

Accessibility: The model contributes to making information more accessible to the deaf community. It allows them to interact with technology and communication devices on an equal footing with the hearing population.

Education and Learning Aid: The project can serve as an effective tool for learning and practicing ASL. It can be integrated into educational settings to aid individuals, including those who are not deaf, in acquiring proficiency in ASL.

Empowerment: Enabling individuals to express themselves through ASL without relying on an interpreter empowers the deaf community. This autonomy is crucial for fostering independence and self-expression.

Disadvantages:

Limited Vocabulary: The model may have limitations in recognizing signs beyond the ASL alphabet. It might not be as effective in capturing the nuances of more complex signs or non-alphabetic gestures.

Variability in Gestures: ASL signs can vary based on factors such as speed, handshape variations, and individual differences. The model may struggle to accurately recognize signs in situations where these variables are prominent.

Environmental Factors: The accuracy of the model may be influenced by environmental factors such as lighting conditions and background clutter. Adverse conditions could impact the model's performance, especially in real-world scenarios.

Hardware Requirements: For real-time applications, the project may have specific hardware requirements. High processing power and efficient cameras may be necessary for optimal performance, potentially limiting accessibility for users with older devices.

Ethical Considerations: The use of technology in communication raises ethical concerns, particularly in terms of privacy and data security. It's crucial to address these concerns and implement safeguards to protect user information.

Cultural Sensitivity: ASL is not a universal language, and there can be cultural variations in sign language. The model may need adaptations to cater to different sign language variants, limiting its applicability in diverse cultural contexts.

11. CONCLUSION

In conclusion, the 'ASL Alphabet Image Recognition' project successfully integrated empathy-driven design, agile development, and advanced technology. Utilizing empathy maps, user stories, and sprints, we created a user-centric solution. The VGG16 model achieved an impressive 95% accuracy, underlining its real-world applicability. The web application, featuring video and image uploads, enhances communication by predicting ASL words in real time. This dynamic approach aligns with our goal of bridging communication gaps between deaf and hearing communities. To improve, future iterations could focus on expanding vocabulary and refining adaptability to diverse signing styles. The project exemplifies the transformative potential of technology in fostering inclusivity and accessibility across communities.

12. FUTURE SCOPE

Looking ahead, the 'ASL Alphabet Image Recognition' project holds promising avenues for future development. Expanding the model's vocabulary to encompass a broader range of ASL signs and refining its adaptability to diverse signing styles can enhance its utility. Additionally, incorporating real-time translation features for conversational ASL holds potential for expanding its impact.

Collaboration with the deaf community for continuous feedback and improvement, along with exploring mobile applications for increased accessibility, could be valuable directions. Integrating cultural adaptations to cater to different sign language variants would make the solution more inclusive on a global scale.

Moreover, exploring opportunities for partnerships with educational institutions could turn the project into a valuable learning tool for ASL acquisition. Ongoing research and development efforts could further optimize the model, making it even more accurate and efficient.

In conclusion, the future scope of the 'ASL Alphabet Image Recognition' project lies in continual refinement, expansion, and collaborative efforts, ensuring its sustained relevance and positive impact in facilitating communication between diverse communities.

13. APPENDIX

Project Demo video link-

https://drive.google.com/file/d/1z_DZ9rAOYhUD6oNnYhxa4wGzYYuqbT7t/view?us p=drive_link

Source Codes -

HTML -

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ASL Recognition</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>

<body>
    <header>
        <div class="header-content">
            <h1>American Sign Language Recognition</h1>
            <p>Explore and learn ASL signs with our interactive recognition tool.</p>
        </div>
    </header>

    <div class="container-wrapper">
        <div class="info-container">
            <div class="info-section">
                <p>The American Sign Language (ASL) is the primary language used by deaf individuals in North America. It is a visual language that uses a combination of hand gestures, facial expressions, and body movements to convey meaning.</p>
            </div class="image-section" >
            <p> Take a look at American Sign Language through the picture below, it contains the alphabets from A-Z, and additionally 3 symbols for space, del, nothing in which J, Z are movements</p>
            <img src="https://www.dictionary.com/e/wp-content/uploads/2018/01/american_sign_language4-790x310.jpg" alt="ASL image">
        </div>

        <div class="predict-container">
            <h3>Getting confused with the ASL 😵‍💫, Use our Deep learning tool to easily predict the ASL sign</h3>
            <button type="button" id="predict-something-button" onclick="showUploadOptions()">Want to predict something?</button>

            <form id="upload-form" action="/predict" method="post" enctype="multipart/form-data">
                <div id="upload-options" style="display: none;">
                    <button type="button" id="video-button" onclick="document.getElementById('video-input').click();">Single Video</button>
                    <input type="file" name="video" id="video-input" accept="video/*" style="display: none;" onchange="previewVideo(this)">
                    <button type="button" id="files-button" onclick="document.getElementById('files-input').click();">Series of Images</button>
                    <input type="file" name="files[]" id="files-input" accept="image/*" multiple style="display: none;" onchange="previewImages(this)">
                </div>
            </form>
            <!--<video id="video-player" style="display:none;"></video>-->
            <!-- Display Images Side by Side -->
            <div id="image-preview" class="flex-container"></div>

            <!-- Predict Button -->
            <button type="button" id="predict-button" onclick="predict(event)">Predict</button>
```

```
            <div id="result"> </div>
        </div>
    </div>
    <script src="{{url_for('static',filename='script.js')}}"></script>

    <input type="file" name="file" id="file-input" accept="image/*" capture="camera"
onchange="previewImage(this, 300)">
</body>
</html>
```

CSS -

```
body {
    font-family: 'Arial', sans-serif;
    background-color: #F1EAFF;
    margin: 0;
    padding: 0;
}

header {
    margin: 10px;
    position: relative;
    text-align: center;
    color: #872341;
    background-color: #FFE3BB;
    padding: 50px;
}

.header-content {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
}

.container-wrapper {
    display: flex;
    justify-content: space-around;
    max-width: 1300px;
    margin: 5px auto;
}

.info-container {
    max-width: 800px;
    margin: 20px auto;
    margin-right:40px;
    text-align: center;
    padding: 20px;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

```css
.predict-container {
    max-width: 400px;
    margin: 20px auto;
    padding: 20px;
    text-align: center;
    background-color: #fff;
    border-radius: 8px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    position: relative;
    max-height: 800px;
    overflow-y: auto;
}

.image-section{
    background-color: #3478db;
    color: #fff;
    padding: 20px;
    border-radius: 8px;
    margin-bottom: 20px;
}

.info-section {
    background-color: #3498db;
    color: #fff;
    padding: 20px;
    border-radius: 8px;
    margin-bottom: 20px;
}

.info-section h2 {
    margin-bottom: 10px;
}

#file-input {
    display: none;
}

#upload-label {
    background-color: #3498db;
    color: #fff;
    padding: 15px 30px;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s;
    margin-bottom: 10px;
}

.upload-label {
    background-color: #3498db;
    color: #fff;
    padding: 15px 30px;
    border-radius: 5px;
    cursor: pointer;
    margin: 5px;
}
```

```
#predict-something-button {
  background-color: #e67e22;
  color: #fff;
  padding: 15px 30px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s;
}

#predict-something-button:hover,
#predict-button:hover {
  background-color: #d35400;
}

#upload-label:hover {
  background-color: #2980b9;
}

#video-button,
#files-button{
  background-color: #3498db;
  color: #fff;
  padding: 15px 30px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s;
  margin: 5px;
}

#predict-button {
  position: absolute;
  bottom: 20px;
  left: 50%;
  transform: translateX(-50%);
  background-color: #e67e22;
  color: #fff;
  padding: 15px 30px;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  transition: background-color 0.3s;
  margin-top:10px;
}

#upload-button:hover,
#predict-button:hover {
  background-color: #d35400;
}

#result {
  position: absolute;
  bottom:1px;
  width: 100%;
  font-weight: bold;
```

```
  text-align: center;
  color: #333;
}
#image-preview {
  margin: 20px auto;
  display: flex;
  justify-content: space-around;
  flex-wrap: wrap;
  position: relative;
}
.flex-container {
  display: flex;
}

.preview-image {
  max-width: 100px;
  margin: 5px;
}

.upload-form {
  display: flex;
  flex-direction: column;
  align-items: flex-end;
}

#upload-instruction {
  margin-top: 10px;
}

.upload-options {
  display: flex;
  flex-direction: column;
  align-items: center;
  margin-top: 10px;
}
.upload-label {
  margin-top: 10px;
}
```

Javascript -

```
document.addEventListener('DOMContentLoaded', function () {
  const videoInput = document.getElementById('video-input');
  const videoPlayer = document.getElementById('video-player');
  const imageContainer = document.getElementById('image-preview');

  videoInput.addEventListener('change', function (event) {
    console.log('Video input change event triggered.');

    const file = event.target.files[0];
    if (file) {
      console.log('Selected file:', file);

      const videoURL = URL.createObjectURL(file);
      videoPlayer.src = videoURL;

      console.log('Video URL:', videoURL);
```

```
            generateImages(videoURL);
        }
    });

    function generateImages(videoURL) {
        console.log('Generating images from video.');

        const video = document.createElement('video');
        video.src = videoURL;
        video.addEventListener('loadedmetadata', function () {
            const duration = video.duration;

            for (let i = 0; i < duration; i++) {
                video.currentTime = i;
                const canvas = document.createElement('canvas');
                const context = canvas.getContext('2d');
                canvas.width = video.videoWidth;
                canvas.height = video.videoHeight;
                context.drawImage(video, 0, 0, canvas.width, canvas.height);

                const img = new Image();
                img.src = canvas.toDataURL('image/png');
                img.alt = `Frame ${i}`;
                imageContainer.appendChild(img);
            }
            console.log('Images generated successfully.');
        });
    }
});
function previewImage(input) {
            const preview = document.getElementById('image-preview');
            preview.innerHTML = '';

            if (input.files && input.files[0]) {
                const reader = new FileReader();

                reader.onload = function (e) {
                    const img = document.createElement('img');
                    img.src = e.target.result;
                    img.style.maxWidth = '100%';
                    preview.appendChild(img);
                };

                reader.readAsDataURL(input.files[0]);
            }
        }

function showUploadOptions() {
    console.log('Upload options are being shown.');
    var uploadOptions = document.getElementById('upload-options');
    uploadOptions.style.display = 'block';
}

function previewImages(input) {
    console.log('Previewing multiple images.');
```

```
    var previewContainer = document.getElementById('image-preview');

    // Clear existing previews
    previewContainer.innerHTML = '';

    var files = input.files;

    for (var i = 0; i < files.length; i++) {
        var file = files[i];
        var reader = new FileReader();

        reader.onload = function (e) {
            var image = document.createElement('img');
            image.src = e.target.result;
            image.className = 'preview-image';
            previewContainer.appendChild(image);
        };

        reader.readAsDataURL(file);
    }
}
function previewVideo(input) {
    const predictContainer = document.querySelector('.predict-container');
    const videoPreview = document.createElement('div');

    // Set styling for the video preview container
    videoPreview.style.width = '100%'; // Set the width to 100%
    videoPreview.style.maxHeight = '400px'; // Set a maximum height
    videoPreview.style.overflow = 'hidden'; // Hide any overflow content

    if (input.files && input.files[0]) {
        const reader = new FileReader();

        reader.onload = function (e) {
            const video = document.createElement('video');
            video.src = e.target.result;
            video.style.width = '100%'; // Set the width to 100%
            video.style.height = 'auto';
            video.controls = true;

            // Append the video element to the video preview container
            videoPreview.appendChild(video);

            // Append the video preview container to the predict container
            predictContainer.appendChild(videoPreview);

            // Automatically remove the video after 1/2 minute (30000 milliseconds)
            setTimeout(function () {
                videoPreview.remove();
            }, 30000);
        };

        reader.readAsDataURL(input.files[0]);
    }
}
```

```
function predictFromVideoFrames() {
    const images = document.getElementById('flex-container').getElementsByTagName('img');
    const formData = new FormData();

    for (let i = 0; i < images.length; i++) {
        const imgDataUrl = images[i].src;
        const blob = dataURLtoBlob(imgDataUrl);
        formData.append('files[]', blob, `frame_${i}.png`);
    }

    fetch('/predict', {
        method: 'POST',
        body: formData
    })
    .then(response => response.json())
    .then(data => {
        document.getElementById('result').innerText = 'Prediction: ' + data.prediction;
    })
    .catch(error => console.error('Error:', error));
}
function dataURLtoBlob(dataURL) {
    const arr = dataURL.split(',');
    const mime = arr[0].match(/:(.*?);/)[1];
    const bstr = atob(arr[1]);
    let n = bstr.length;
    const u8arr = new Uint8Array(n);
    while (n--) {
        u8arr[n] = bstr.charCodeAt(n);
    }
    return new Blob([u8arr], { type: mime });
}
function predict(event) {
    event.preventDefault();  // Prevent the default form submission behavior

    const form = document.getElementById('upload-form');
    const resultElement = document.getElementById('result');

    const formData = new FormData(form);

    console.log('Form data:', formData);

    if (formData.has('files[]')) {
        // For multiple files
        console.log('Processing multiple files.');
        fetch('/predict', {
            method: 'POST',
            body: formData,
        })
        .then(response => response.json())
        .then(data => {
            console.log('Prediction data:', data); // Log the prediction data
            resultElement.innerText = 'Prediction: ' + data.prediction;
        })
        .catch(error => console.error('Error:', error));
    } else if (formData.has('file')) {
        // For a single file
```

```
        console.log('Processing a single file.');
        fetch('/predict', {
          method: 'POST',
          body: formData,
        })
        .then(response => response.json())
        .then(data => {
          console.log('Prediction data:', data); // Log the prediction data
          resultElement.innerText = 'Prediction: ' + data.prediction;
        })
        .catch(error => console.error('Error:', error));
      } else if (formData.has('video')) {
        // For video file
        console.log('Processing video file.');
        predictFromVideoFrames(formData);
      } else {
        console.error('No file(s) found in the FormData.');
      }
    }
```

Python(Flask application)-

```
from flask import Flask, render_template, request, jsonify
from tensorflow.keras.models import load_model
from PIL import Image
import numpy as np
from werkzeug.utils import secure_filename
import os

app = Flask(__name__)

# Load your model
model = load_model('weights.h5', compile=False)  # Update with your actual path

# Define the allowed extensions for file uploads
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

# Your existing Python code

def predict_image(file_path):
    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', 'del', 'nothing', 'space']

    # Process the image for prediction (you might need to resize, normalize, etc.)
    img = Image.open(file_path)
    img = img.resize((32, 32))  # Adjust the size according to your model's input shape
    img_array = np.array(img) / 255.0  # Normalize
    img_array = img_array[:,:,:3]
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension

    # Make prediction
    prediction = model.predict(img_array)

    predicted_class = labels[np.argmax(prediction)]
```

```python
    print('Predicted class:', predicted_class)  # Log the predicted class

    return predicted_class

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        if 'files[]' not in request.files:
            return jsonify({'error': 'No file part'})

        files = request.files.getlist('files[]')
        print('Number of files:', len(files))  # Log the number of files
        print('Received files:', request.files)


        if len(files) == 1:  # Single image prediction
            file = files[0]

            if file.filename == '':
                return jsonify({'error': 'No selected file'})

            if file and allowed_file(file.filename):
                filename = secure_filename(file.filename)
                file_path = os.path.join('uploads', filename)
                file.save(file_path)

                predicted_class = predict_image(file_path)
                return jsonify({'prediction': f'Your image represents {predicted_class}'})

        elif len(files) > 1:  # Multiple images prediction
            predictions = []

            for i, file in enumerate(files):
                if file and allowed_file(file.filename):
                    filename = secure_filename(file.filename)
                    file_path = os.path.join('uploads', f'{i}_{filename}')  # Add an index as a prefix
                    file.save(file_path)

                    predicted_class = predict_image(file_path)
                    predictions.append(predicted_class)

            predicted_word = ''.join(predictions)

            return jsonify({'prediction': f'Your images represent {predicted_word}'})

if __name__ == '__main__':
    app.run(debug=False, threaded=False)
```

Python(Deep learning model)-

```
!mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
```

```
! chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d grassknoted/asl-alphabet

!unzip asl-alphabet.zip -d asl-alphabet

# Load Data
import os
import cv2
import numpy as np

# Data Visualisation
import matplotlib.pyplot as plt

# Model Training
from tensorflow.keras import utils
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D, BatchNormalization
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import VGG16

# Warning
import warnings
warnings.filterwarnings("ignore")

# Main
import os
import glob
import cv2
import numpy as np
import pandas as pd
import gc
import string
import time
import random
from PIL import Image
from tqdm import tqdm
tqdm.pandas()

# Visualization
import matplotlib
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# Model
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img, img_to_array, array_to_img
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten, Dropout, GlobalAveragePooling2D
from keras.models import load_model, Model
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping
from sklearn.metrics import classification_report
```

```
# Configuration
class CFG:
    # Set the batch size for training
    batch_size = 128
    # Set the height and width of input images
    img_height = 32
    img_width = 32
    epochs = 10
    num_classes = 29
    # Define the number of color channels in input images
    img_channels = 3
# Define a function to set random seeds for reproducibility
def seed_everything(seed: int):
    random.seed(seed)
    # Set the environment variable for Python hash seed
    os.environ["PYTHONHASHSEED"] = str(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)

# Labels
TRAIN_PATH = "/content/asl-alphabet/asl_alphabet_train/asl_alphabet_train"
labels = []
# Generate a list of uppercase letters in the English alphabet
alphabet = list(string.ascii_uppercase)
labels.extend(alphabet)
# Add special labels for 'delete', 'nothing', and 'space' gestures
labels.extend(["del", "nothing", "space"])
print(labels)

# Create Metadata
list_path = []
list_labels = []
for label in labels:
    # Create a path pattern to match all image files for the current label
    label_path = os.path.join(TRAIN_PATH, label, "*")
    # Use glob to retrieve a list of image file paths that match the pattern
    image_files = glob.glob(label_path)
    sign_label = [label] * len(image_files)
    list_path.extend(image_files)
    list_labels.extend(sign_label)
metadata = pd.DataFrame({
    "image_path": list_path,
    "label": list_labels
})
metadata

# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    metadata['image_path'],
    metadata['label'],
    test_size=0.2,
    random_state=2253,
    shuffle=True,
    stratify=metadata['label']
)
```

```
# Create a DataFrame for the training set test set
data_train = pd.DataFrame({
    'image_path': X_train,
    'label': y_train
})

data_test = pd.DataFrame({
    'image_path': X_test,
    'label': y_test
})

# Split the training set into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    data_train['image_path'],
    data_train['label'],
    test_size=0.2/0.7,  # Assuming you want 20% for validation out of the training set
    random_state=2253,
    shuffle=True,
    stratify=data_train['label']
)

# Create a DataFrame for the validation set
data_val = pd.DataFrame({
    'image_path': X_val,
    'label': y_val
})

def data_augmentation():
    datagen = ImageDataGenerator(
        rescale=1/255.,
        # Add other augmentation parameters as needed
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest'
    )
    train_generator = datagen.flow_from_dataframe(
        data_train,
        directory='./',
        x_col='image_path',
        y_col='label',
        class_mode='categorical',
        batch_size=CFG.batch_size,
        target_size=(CFG.img_height, CFG.img_width)
    )
    validation_generator = datagen.flow_from_dataframe(
        data_val,
        directory='./',
        x_col='image_path',
        y_col='label',
        class_mode='categorical',
        batch_size=CFG.batch_size,
```

```
        target_size=(CFG.img_height, CFG.img_width)
    )

    test_generator = datagen.flow_from_dataframe(
        data_test,  # Assuming you have a DataFrame for test data
        directory='./',
        x_col='image_path',
        y_col='label',
        class_mode='categorical',
        batch_size=CFG.batch_size,
        target_size=(CFG.img_height, CFG.img_width),
        shuffle=False  # Set to False for test data
    )

    return train_generator, validation_generator, test_generator

# Seed for reproducibility
seed_everything(2253)

# Get the generators
train_generator, validation_generator, test_generator = data_augmentation()

# Define input shape
input_shape = (32, 32, 3)

# Load the VGG16 model without the top (classification) layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=input_shape)

# Add your custom classification layers on top of the base model
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(128, activation='relu')(x)  # You can adjust the number of units as needed
predictions = Dense(29, activation='softmax')(x)  # num_classes is the number of classes in your dataset

# Create the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Summarize the model architecture
model.summary()


# Compile the model
model.compile(optimizer=Adam(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

# Create a ModelCheckpoint callback
checkpoint_callback = ModelCheckpoint(
    filepath='/content/sample_data/best_model_weights.h5',
    monitor='val_accuracy',  # Monitor validation accuracy for saving the best model
    save_best_only=True,
    mode='max',
    verbose=1
)
# Train the model using the fit method
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // CFG.batch_size,  # Number of steps per epoch
    epochs=CFG.epochs,  # Number of training epochs
```

```
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // CFG.batch_size,  # Number of validation steps
    callbacks=[checkpoint_callback],
    shuffle=True,
    verbose=1
)


scores = model.evaluate(test_generator)
print("%s: %2f%%" % ("Evaluate Test Accuracy", scores[1]*100))

# Confusion Matrix
fine_tuned_model = load_model("/content/sample_data/best_model_weights.h5")
predictions = fine_tuned_model.predict(test_generator)

# Get the true labels from the generator
true_labels = test_generator.classes

# Compute the confusion matrix using tf.math.confusion_matrix
confusion_matrix = tf.math.confusion_matrix(
    labels = true_labels,
    predictions = predictions.argmax(axis=1),
    num_classes = 29
)

#Classification report
predictions = model.predict(test_generator)
predicted_labels = np.argmax(predictions, axis=1)
true_labels = test_generator.classes
report = classification_report(true_labels, predicted_labels, target_names=labels)
print(report)

# Load the saved model
model = tf.keras.models.load_model('/content/sample_data/best_model_weights.h5')

# Testing with an image
image_path = '/content/asl-alphabet/asl_alphabet_train/asl_alphabet_train/Y/Y10.jpg'
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img,(32,32))

img = tf.keras.applications.mobilenet_v2.preprocess_input(img)

# Predict the class of the image
predictions = model.predict(np.array([img]))

# Get the class with the highest probability
predicted_class = labels[np.argmax(predictions)]

print(f"The image is predicted to belong to class: {predicted_class}")
```