Project Report Documentation

1. INTRODUCTION

1.1 Project Overview

In a world driven by technological advancements, the potential for AI to bridge communication gaps is both promising and transformative. American Sign Language (ASL) stands as a vital mode of communication for the deaf and hard of hearing community. However, the accessibility of ASL relies heavily on the comprehension and interpretation of its intricate gestures.

This project focuses on leveraging the power of image recognition technology to decode and understand ASL alphabetic signs through visual data. The aim is to create an innovative system capable of accurately identifying and interpreting ASL alphabet gestures from images or video input.

The significance of this endeavor lies not only in facilitating smoother communication but also in fostering inclusivity and accessibility for individuals whose primary mode of expression is ASL. By harnessing machine learning algorithms and image recognition techniques, this project endeavors to break down barriers and enhance the way we perceive and interact with ASL.

Throughout this overview, we will delve into the methodologies, challenges, and anticipated outcomes of this ASL Alphabetic Image Recognition project, highlighting its potential to revolutionize communication and inclusivity for the deaf and hard of hearing community.

1.2 Purpose

In the realm of communication accessibility, the development of ASL Alphabetic Image Recognition stands as a pioneering initiative. American Sign Language, a profound mode of communication for the Deaf community, relies heavily on gestures, expressions, and fingerspelling. However, the integration of technology to interpret and translate these gestures into text has seen substantial progress, yet the recognition of fingerspelled alphabets within ASL remains a challenging frontier.

ASL Alphabetic Image Recognition serves as a technological bridge, aiming to break barriers by accurately identifying and transcribing the intricate gestures of ASL fingerspelling. Its purpose is multifaceted: empowering individuals within the Deaf community by facilitating seamless communication, fostering inclusivity in technological advancements, and promoting a deeper understanding and integration of ASL within the broader communication landscape.

This innovation not only strives for functional accuracy but also embodies a spirit of empowerment and inclusion. By harnessing the potential of machine learning, computer vision, and pattern recognition, ASL Alphabetic Image Recognition endeavors to revolutionize the accessibility and integration of American Sign Language in our interconnected world.

2. LITERATURE SURVEY

2.1 Existing problem

ASL fingerspelling faces challenges due to hand gesture complexity, variability, context sensitivity, limited datasets, real-time processing, and segmentation. Accurate recognition requires user-friendly systems that can adapt to various signing styles and ensure a positive user experience for wider adoption.

2.2 References

"DeepASL: Enabling Ubiquitous and Non-Intrusive Word and Alphabet Gesture Recognition for ASL Using Deep Learning" (2018) by Abdullah-Al-Zubaer Imran, et al.

- This paper explores the application of deep learning techniques for recognizing ASL alphabetic gestures, providing insights into convolutional neural networks' effectiveness in this domain.

"Fingerspelling Recognition in American Sign Language Videos Using Convolutional Neural Networks" (2019) by Mihai Gabriel Constantin and Radu Tudor Ionescu.

- This research delves into the utilization of CNNs for fingerspelling recognition in ASL videos, presenting methodologies and experimental results for improved recognition accuracy.

"Real-Time American Sign Language Alphabet Recognition Using Convolutional Neural Networks" (2020) by Azim Sekandarzad, et al.

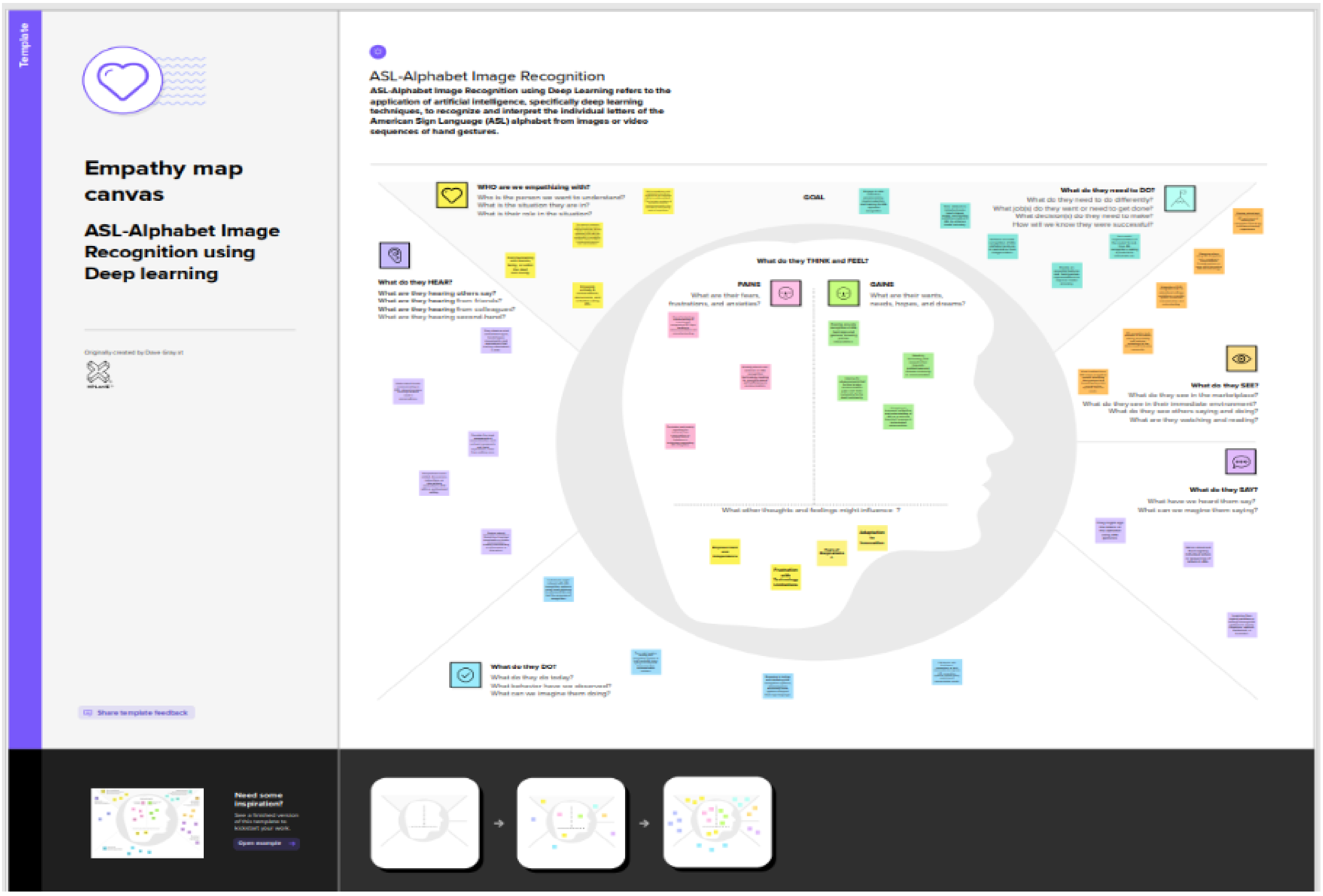
- The study focuses on real-time recognition of the ASL alphabet through CNN-based models, addressing the challenges and potential solutions for accurate and efficient recognition.

2.3 Problem Statement Definition

Research on sign language recognition techniques, accuracy, performance metrics, datasets, real-time challenges, user-centric studies, context and grammar incorporation, and AI and deep learning advancements are essential for improving accuracy and robustness in ASL fingerspelling recognition.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



Empathy map canvas

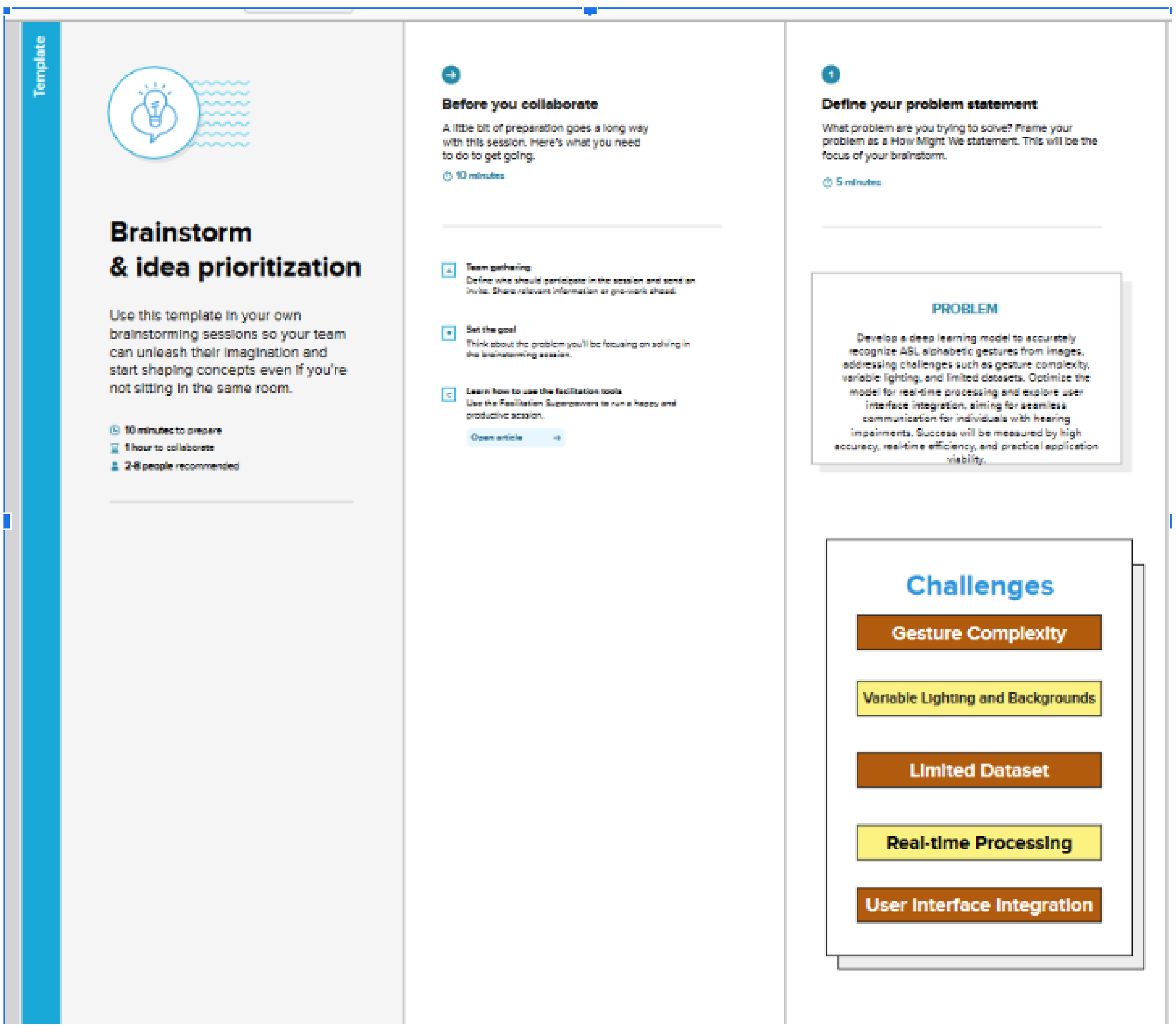
ASL-Alphabet Image Recognition using Deep learning

ASL-Alphabet Image Recognition

ASL-Alphabet Image Recognition using Deep Learning refers to the application of artificial intelligence, specifically deep learning techniques, to recognize and interpret the individual letters of the American Sign Language (ASL) alphabet from images or video sequences of hand gestures.

Template
Originally created by Dave Gray at
WHO are we empathizing with?
Who is the person we want to understand?
What is the situation they are in?
What is their role in the situation?
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?
What do they HEAR?
What are they hearing others say?
What are they hearing from friends?
What are they hearing from colleagues?
What are they hearing second-hand?
What do they THINK and FEEL?
PAINS
What are their fears, frustrations, and anxieties?
GAINS
What are their wants, needs, hopes, and dreams?
What other thoughts and feelings might influence ?
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?
What do they DO?
What do they do today?
What behavior have we observed?
What can we imagine them doing?
Share template feedback
Need some inspiration?
Open example

3.2 Ideation & Brainstorming



Template

Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

- 10 minutes to prepare
- 1 hour to collaborate
- 2-8 people recommended

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

Team gathering
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

Set the goal
Think about the problem you'll be focusing on solving in the brainstorming session.

Learn how to use the facilitation tools
Use the Facilitation Superpowers to run a happy and productive session.

Open article →

Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

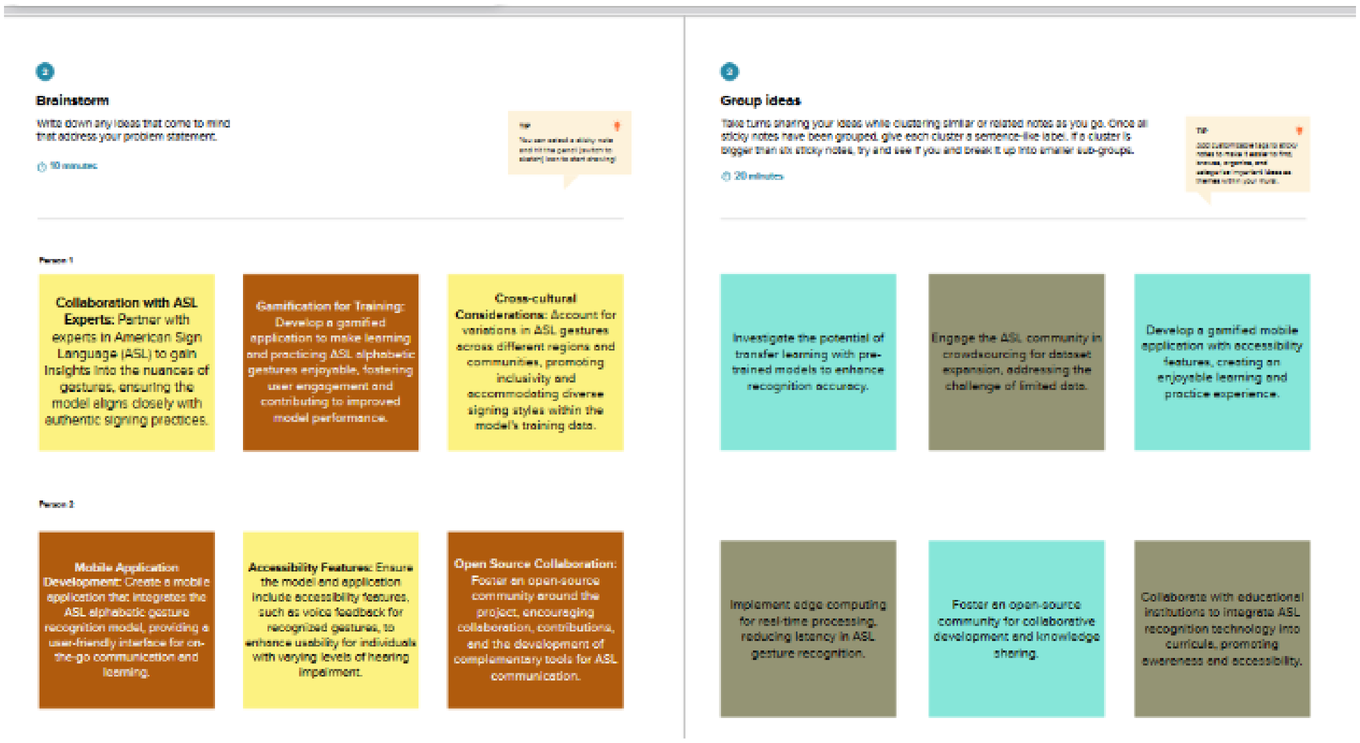
5 minutes

PROBLEM

Develop a deep learning model to accurately recognize ASL alphabetic gestures from images, addressing challenges such as gesture complexity, variable lighting, and limited datasets. Optimize the model for real-time processing and explore user interface integration, aiming for seamless communication for individuals with hearing impairments. Success will be measured by high accuracy, real-time efficiency, and practical application viability.

Challenges

- Gesture Complexity
- Variable Lighting and Backgrounds
- Limited Dataset
- Real-time Processing
- User Interface Integration



2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Person 1

Collaboration with ASL Experts: Partner with experts in American Sign Language (ASL) to gain insights into the nuances of gestures, ensuring the model aligns closely with authentic signing practices.

Gamification for Training: Develop a gamified application to make learning and practicing ASL alphabetic gestures enjoyable, fostering user engagement and contributing to improved model performance.

Cross-cultural Considerations: Account for variations in ASL gestures across different regions and communities, promoting inclusivity and accommodating diverse signing styles within the model's training data.

Person 2

Mobile Application Development: Create a mobile application that integrates the ASL alphabetic gesture recognition model, providing a user-friendly interface for on-the-go communication and learning.

Accessibility Features: Ensure the model and application include accessibility features, such as voice feedback for recognized gestures, to enhance usability for individuals with varying levels of hearing impairment.

Open Source Collaboration: Foster an open-source community around the project, encouraging collaboration, contributions, and the development of complementary tools for ASL communication.

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

Investigate the potential of transfer learning with pre-trained models to enhance recognition accuracy.

Engage the ASL community in crowdsourcing for dataset expansion, addressing the challenge of limited data.

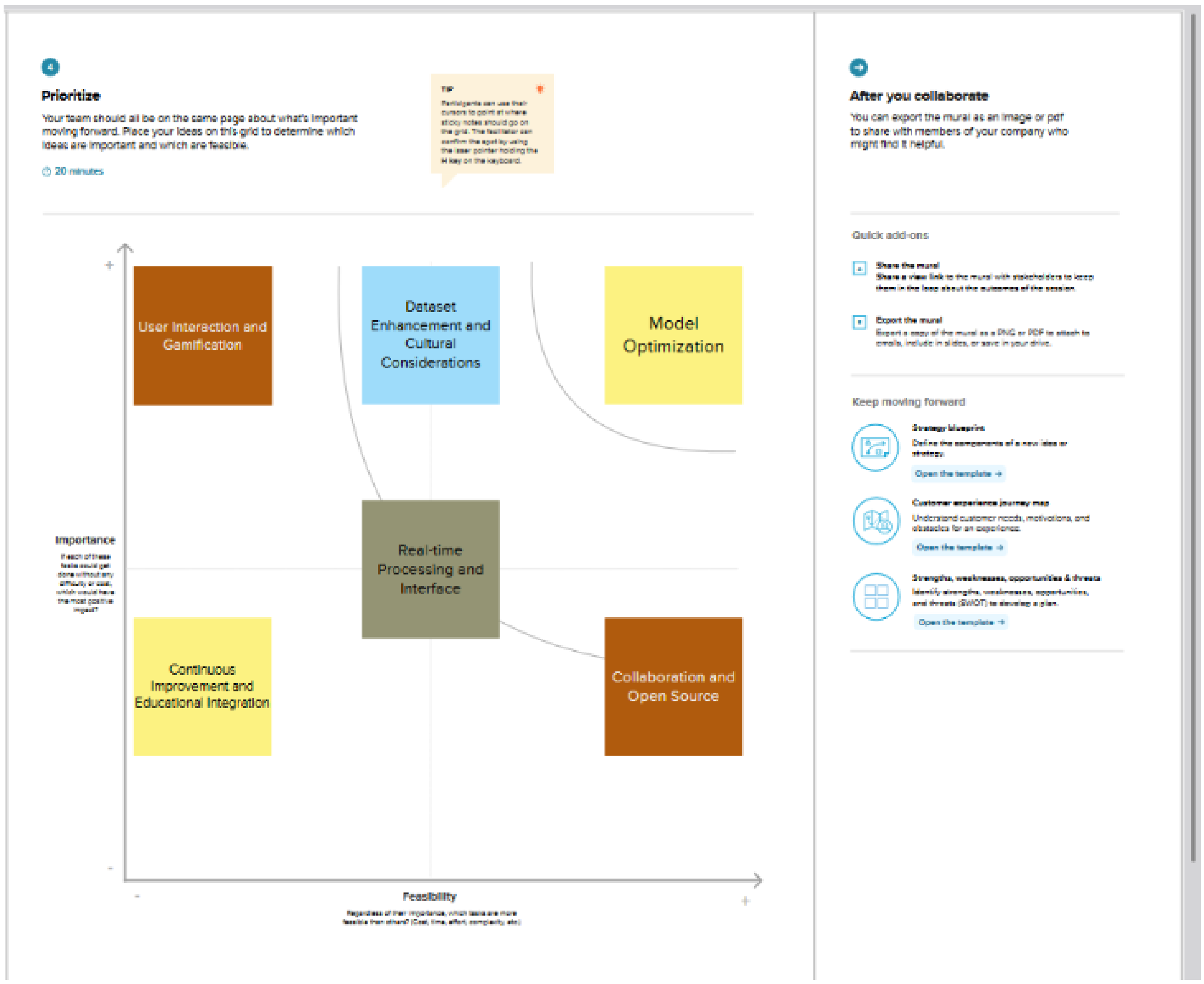
Develop a gamified mobile application with accessibility features, creating an enjoyable learning and practice experience.

Implement edge computing for real-time processing, reducing latency in ASL gesture recognition.

Foster an open-source community for collaborative development and knowledge sharing.

Collaborate with educational institutions to integrate ASL recognition technology into curricula, promoting awareness and accessibility.

Step-3: Idea Prioritization



4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the H key on the keyboard.

+
User Interaction and Gamification
Dataset Enhancement and Cultural Considerations
Model Optimization
Importance
If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?
Real-time Processing and Interface
Continuous Improvement and Educational Integration
Collaboration and Open Source
-
-
Feasibility
Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)
+

After you collaborate

You can export the mural as an image or pdf to share with members of your company who might find it helpful.

Quick add-ons

Share the mural
Share a view link to the mural with stakeholders to keep them in the loop about the outcomes of the session.

Export the mural
Export a copy of the mural as a PNG or PDF to attach to emails, include in slides, or save in your drive.

Keep moving forward

Strategy blueprint
Define the components of a new idea or strategy.
Open the template →

Customer experience journey map
Understand customer needs, motivations, and obstacles for an experience.
Open the template →

Strengths, weaknesses, opportunities & threats
Identify strengths, weaknesses, opportunities, and threats (SWOT) to develop a plan.
Open the template →

4. REQUIREMENT ANALYSIS

4.1 Functional requirement

1. **Image Input Recognition:**
    - The system must accurately detect and interpret ASL fingerspelling alphabets from image inputs.
2. **Alphabet Classification:**
    - It should classify each recognized alphabet accurately, converting it into its corresponding textual representation.
3. **Real-Time Processing:**
    - The system should operate in near-real-time, swiftly recognizing and transcribing ASL alphabets to ensure seamless communication.
4. **Multiple Alphabets Support:**
    - Capable of recognizing a wide range of ASL fingerspelling alphabets, including variations in hand shapes, positions, and movements.
5. **Scalability and Adaptability:**
    - Ability to adapt and learn from new datasets to continuously improve accuracy and recognize diverse signing styles.

4.2 Non-Functional requirements

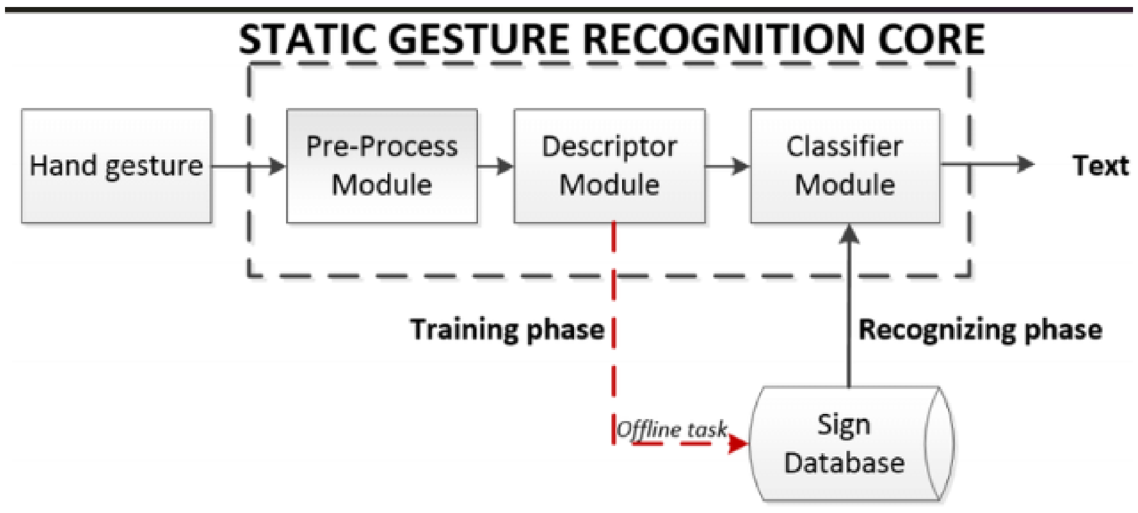
1. **Accuracy and Reliability:**
    - High accuracy in alphabet recognition to ensure reliable translation and minimize errors in transcription.
2. **Robustness and Stability:**
    - The system should perform consistently across various lighting conditions, hand orientations, and backgrounds to ensure robust functionality.
3. **Security and Privacy:**
    - Data privacy measures must be in place to protect any stored or transmitted personal information of users.
4. **Usability and Accessibility:**
    - The user interface should be intuitive and accessible, ensuring ease of use for individuals within the Deaf community.
5. **Performance Efficiency:**

    - Optimal performance without excessive computational requirements, ensuring swift processing and response times.
6. **Adherence to Standards:**
    - Compliance with ASL linguistic and cultural norms to accurately represent and interpret the language's nuances.
7. **Compatibility and Integration:**
    - Integration capabilities with different devices or platforms to promote widespread usage and accessibility.
8. **Documentation and Support:**
    - Comprehensive documentation and support services for developers and end-users to facilitate understanding and troubleshooting.

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.



User Stories

Use the below template to list all the user stories for the ASL - Alphabetic Image Recognition

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
As a language learner	Recognition of ASL alphabetic signs through image input	USN-1	As a user, I want to be able to take a photo of an ASL alphabetic sign using the app's camera feature.	The app should feature a camera interface for users to capture clear, focused images, and prompt them to confirm before processing.	High	Sprint-1
As a parent of a child with hearing impairments	Interactive learning experience for ASL alphabets	USN-2	As a parent, I want the app to have interactive games or quizzes to help my child learn and practice ASL alphabets.	The app should feature interactive, engaging, and intuitive games for children learning ASL alphabets, with progress tracking to track their learning journey.	Medium	Sprint-1
As a teacher of a deaf education class	Comprehensive ASL dictionary integration	USN-3	As a teacher, I want the app to have an extensive ASL dictionary where I can search for signs and access their meanings, usage, and variations.	The app should offer a searchable database of ASL signs, complete with video demonstrations, descriptions, and usage examples, allowing users to easily navigate and access information.	High	Sprint-2
As a developer integrating the ASL recognition API	API documentation and integration support	USN-4	As a developer, I want clear and comprehensive documentation for the ASL recognition API, including sample code and integration guidelines.	The API documentation should cover endpoints, parameters, and response formats, provide sample code snippets for popular programming languages, and offer support resources like forums or	Medium	Sprint-2

5.2 Solution Architecture

Designing a solution architecture for ASL (American Sign Language) alphabet image recognition using deep learning involves several key components.IT involves designing a system to recognize and interpret the hand gestures representing individual letters in ASL.

Steps involved:

1. Data Collection and Preprocessing:

- **Dataset Acquisition**
- **Data Preprocessing**

2. Model Selection:

- **Convolutional Neural Network (CNN)**

3. Model Training:

- **Training Setup**
- **Transfer Learning**

4. Deployment:

- **Model Deployment**
- **User Interface (UI)**

5. Continuous Improvement

- **Monitoring and Feedback**

Considerations:

- **Data Privacy and Security**
- **Scalability**
- **Model Interpretability**

Technology Stack

- **Programming Languages**
- **Libraries and Frameworks**
- **Deployment**

Example - Solution Architecture Diagram:

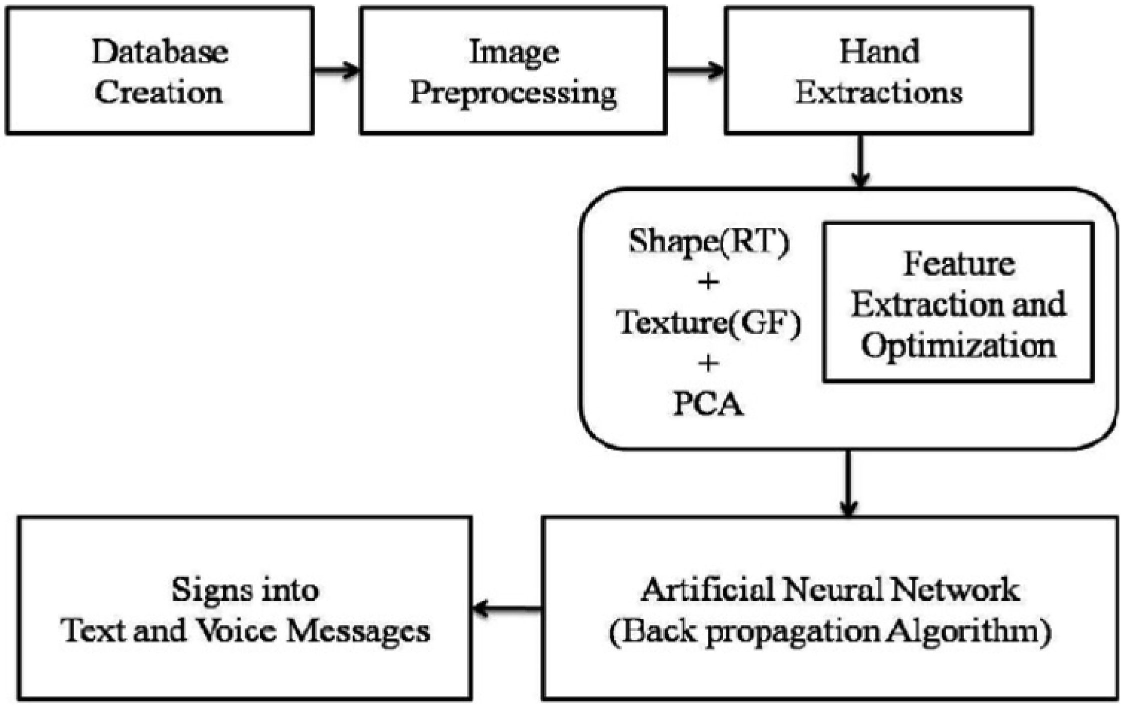
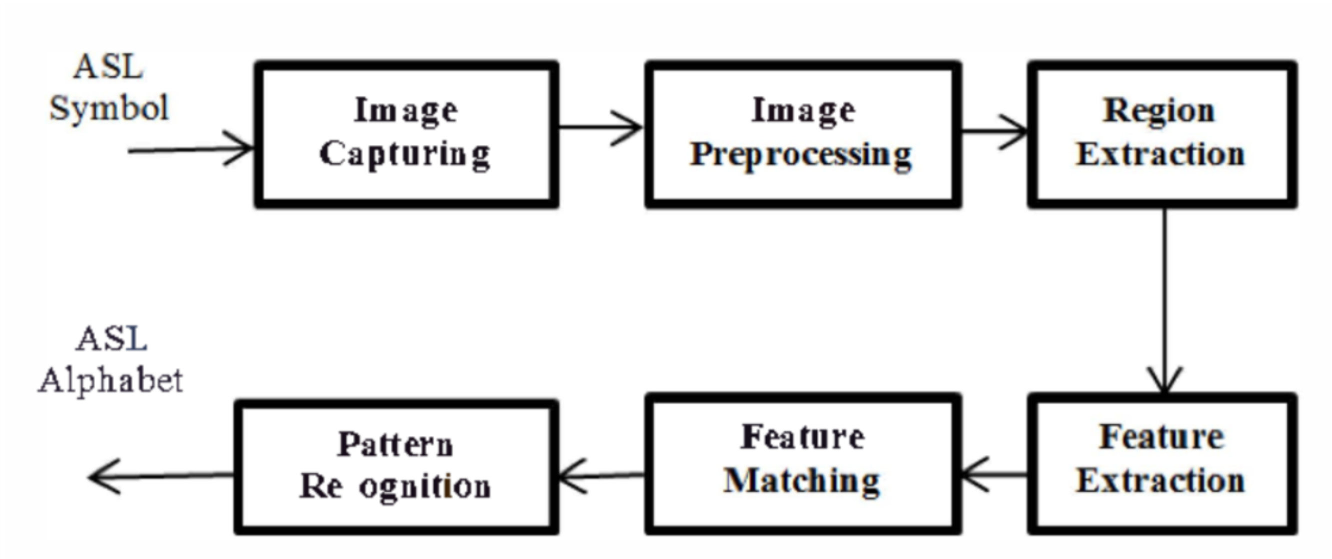

Database Creation
Image Preprocessing
Hand Extractions
Shape(RT) + Texture(GF) + PCA
Feature Extraction and Optimization
Artificial Neural Network (Back propagation Algorithm)
Signs into Text and Voice Messages

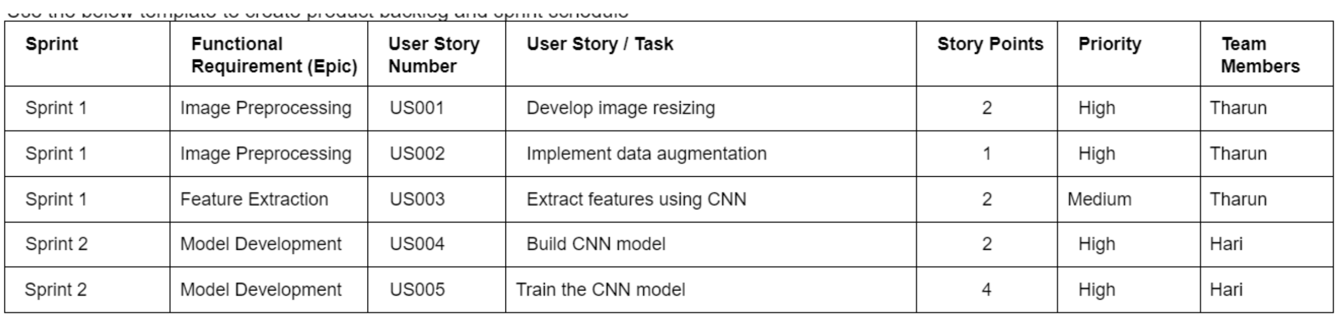
Figure 1: Architecture and data flow of the ASL- Alphabet Image Recognition

6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

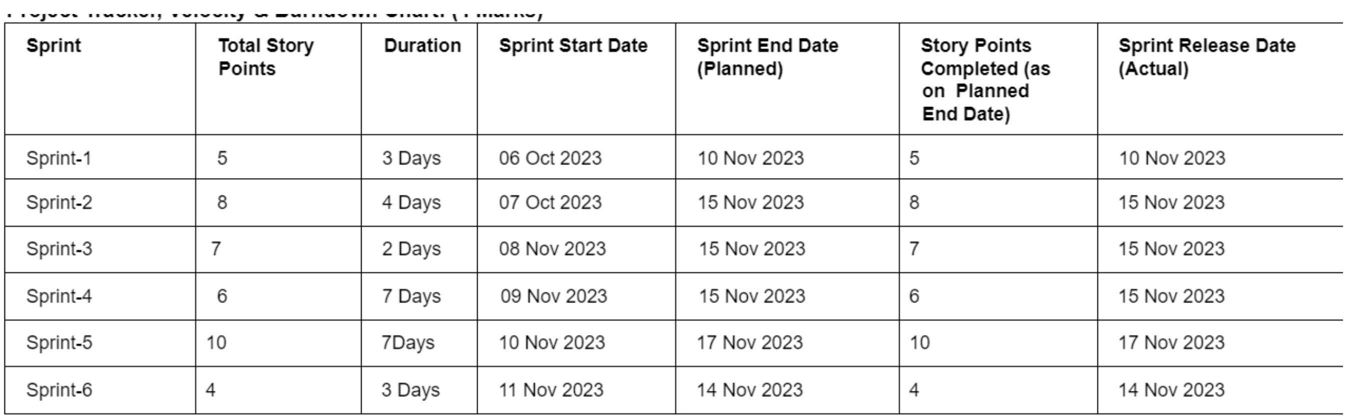


6.2 Sprint Planning & Estimation


Use the below template to create product backlog and sprint schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint 1	Image Preprocessing	US001	Develop image resizing	2	High	Tharun
Sprint 1	Image Preprocessing	US002	Implement data augmentation	1	High	Tharun
Sprint 1	Feature Extraction	US003	Extract features using CNN	2	Medium	Tharun
Sprint 2	Model Development	US004	Build CNN model	2	High	Hari
Sprint 2	Model Development	US005	Train the CNN model	4	High	Hari

6.3 Sprint Delivery Schedule


Project Tracker, Velocity & Burndown Chart: (4 Marks)

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	5	3 Days	06 Oct 2023	10 Nov 2023	5	10 Nov 2023
Sprint-2	8	4 Days	07 Oct 2023	15 Nov 2023	8	15 Nov 2023
Sprint-3	7	2 Days	08 Nov 2023	15 Nov 2023	7	15 Nov 2023
Sprint-4	6	7 Days	09 Nov 2023	15 Nov 2023	6	15 Nov 2023
Sprint-5	10	7Days	10 Nov 2023	17 Nov 2023	10	17 Nov 2023
Sprint-6	4	3 Days	11 Nov 2023	14 Nov 2023	4	14 Nov 2023

7. CODING & SOLUTIONING (Explain the features added in the project along with code)

7.1 Feature 1

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

7.2 Feature2

The ASL Alphabetic Image Recognition system uses various techniques to enhance its performance.

These include data preprocessing, CNN architecture, transfer learning, gesture localization, real-time prediction, an intuitive user interface, model evaluation, deployment, accessibility features, and error handling.

Data preprocessing involves transforming raw image data into a suitable format, CNN extracts features, transfer learning fine-tuning on limited datasets, and implementing techniques for hand or gesture identification.

Real-time prediction is achieved through capturing live video input and processing it.

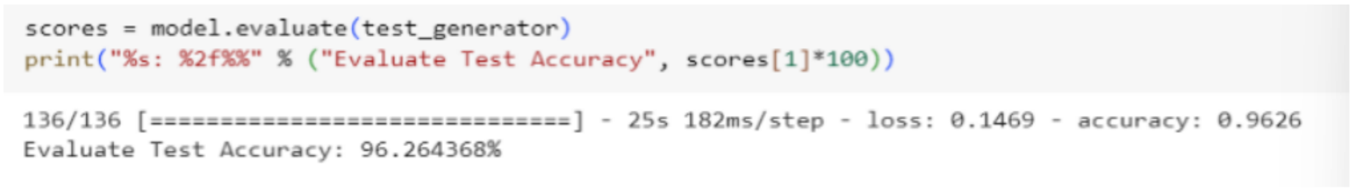
The system also includes an intuitive UI, accessibility features, and robust error handling mechanisms.

8. PERFORMANCE TESTING

8.1 Performance Metrics

1.Accuracy:- We have got training and testing accuracy as follows:- Training accuracy:- 94.98%

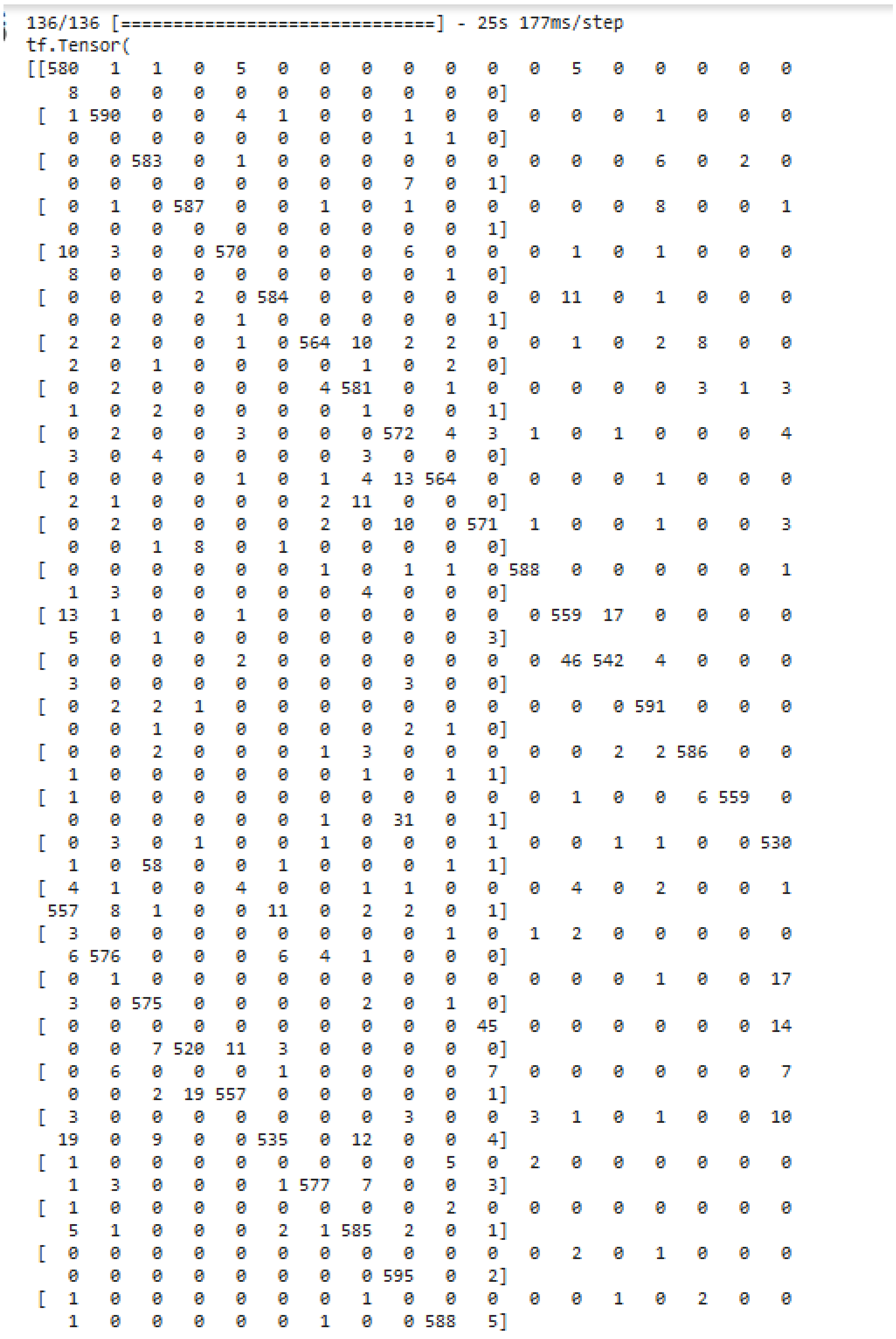
Testing accuracy:- 96.26%



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

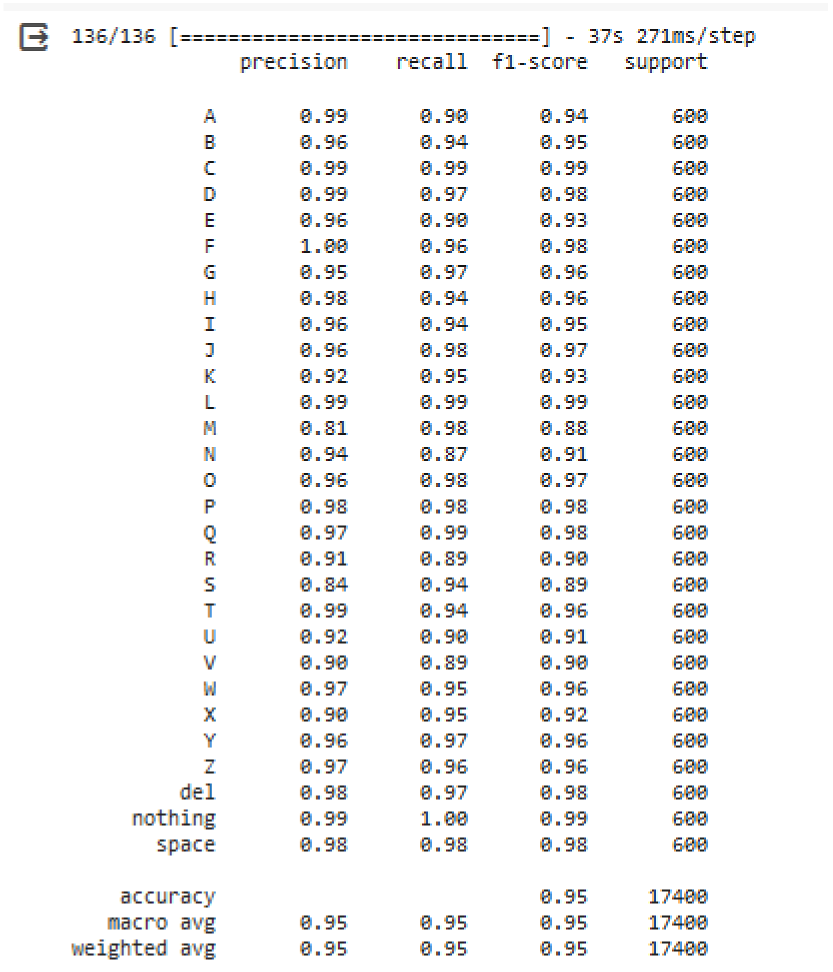
```
136/136 [==============================] - 25s 182ms/step - loss: 0.1469 - accuracy: 0.9626
Evaluate Test Accuracy: 96.264368%
```

2.Confusion Matrix:



```
136/136 [==============================] - 25s 177ms/step
tf.Tensor(
[[580   1   1   0   5   0   0   0   0   0   0   0   5   0   0   0   0   0
    8   0   0   0   0   0   0   0   0   0   0]
 [  1 590   0   0   4   1   0   0   1   0   0   0   0   0   1   0   0   0
    0   0   0   0   0   0   0   0   1   1   0]
 [  0   0 583   0   1   0   0   0   0   0   0   0   0   0   6   0   2   0
    0   0   0   0   0   0   0   0   7   0   1]
 [  0   1   0 587   0   0   1   0   1   0   0   0   0   0   8   0   0   1
    0   0   0   0   0   0   0   0   0   0   1]
 [ 10   3   0   0 570   0   0   0   6   0   0   0   1   0   1   0   0   0
    8   0   0   0   0   0   0   0   0   1   0]
 [  0   0   0   2   0 584   0   0   0   0   0   0  11   0   1   0   0   0
    0   0   0   0   1   0   0   0   0   0   1]
 [  2   2   0   0   1   0 564  10   2   2   0   0   1   0   2   8   0   0
    2   0   1   0   0   0   0   1   0   2   0]
 [  0   2   0   0   0   0   4 581   0   1   0   0   0   0   0   3   1   3
    1   0   2   0   0   0   0   1   0   0   1]
 [  0   2   0   0   3   0   0   0 572   4   3   1   0   1   0   0   0   4
    3   0   4   0   0   0   0   3   0   0   0]
 [  0   0   0   0   1   0   1   4  13 564   0   0   0   0   1   0   0   0
    2   1   0   0   0   0   2  11   0   0   0]
 [  0   2   0   0   0   0   2   0  10   0 571   1   0   0   1   0   0   3
    0   0   1   8   0   1   0   0   0   0   0]
 [  0   0   0   0   0   0   1   0   1   1   0 588   0   0   0   0   0   1
    1   3   0   0   0   0   0   4   0   0   0]
 [ 13   1   0   0   1   0   0   0   0   0   0   0 559  17   0   0   0   0
    5   0   1   0   0   0   0   0   0   0   3]
 [  0   0   0   0   2   0   0   0   0   0   0   0  46 542   4   0   0   0
    3   0   0   0   0   0   0   0   3   0   0]
 [  0   2   2   1   0   0   0   0   0   0   0   0   0   0 591   0   0   0
    0   0   1   0   0   0   0   0   2   1   0]
 [  0   0   2   0   0   0   1   3   0   0   0   0   0   2   2 586   0   0
    1   0   0   0   0   0   0   1   0   1   1]
 [  1   0   0   0   0   0   0   0   0   0   0   0   1   0   0   6 559   0
    0   0   0   0   0   0   1   0  31   0   1]
 [  0   3   0   1   0   0   1   0   0   0   1   0   0   1   1   0   0 530
    1   0  58   0   0   1   0   0   0   1   1]
 [  4   1   0   0   4   0   0   1   1   0   0   0   4   0   2   0   0   1
  557   8   1   0   0  11   0   2   2   0   1]
 [  3   0   0   0   0   0   0   0   0   1   0   1   2   0   0   0   0   0
    6 576   0   0   0   6   4   1   0   0   0]
 [  0   1   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  17
    3   0 575   0   0   0   0   2   0   1   0]
 [  0   0   0   0   0   0   0   0   0   0  45   0   0   0   0   0   0  14
    0   0   7 520  11   3   0   0   0   0   0]
 [  0   6   0   0   0   1   0   0   0   0   7   0   0   0   0   0   0   7
    0   0   2  19 557   0   0   0   0   0   1]
 [  3   0   0   0   0   0   0   0   3   0   0   3   1   0   1   0   0  10
   19   0   9   0   0 535   0  12   0   0   4]
 [  1   0   0   0   0   0   0   0   0   5   0   2   0   0   0   0   0   0
    1   3   0   0   0   1 577   7   0   0   3]
 [  1   0   0   0   0   0   0   0   0   2   0   0   0   0   0   0   0   0
    5   1   0   0   0   2   1 585   2   0   1]
 [  0   0   0   0   0   0   0   0   0   0   0   0   2   0   1   0   0   0
    0   0   0   0   0   0   0   0 595   0   2]
 [  1   0   0   0   0   0   0   1   0   0   0   0   0   1   0   2   0   0
    1   0   0   0   0   0   1   0   0 588   5]
```

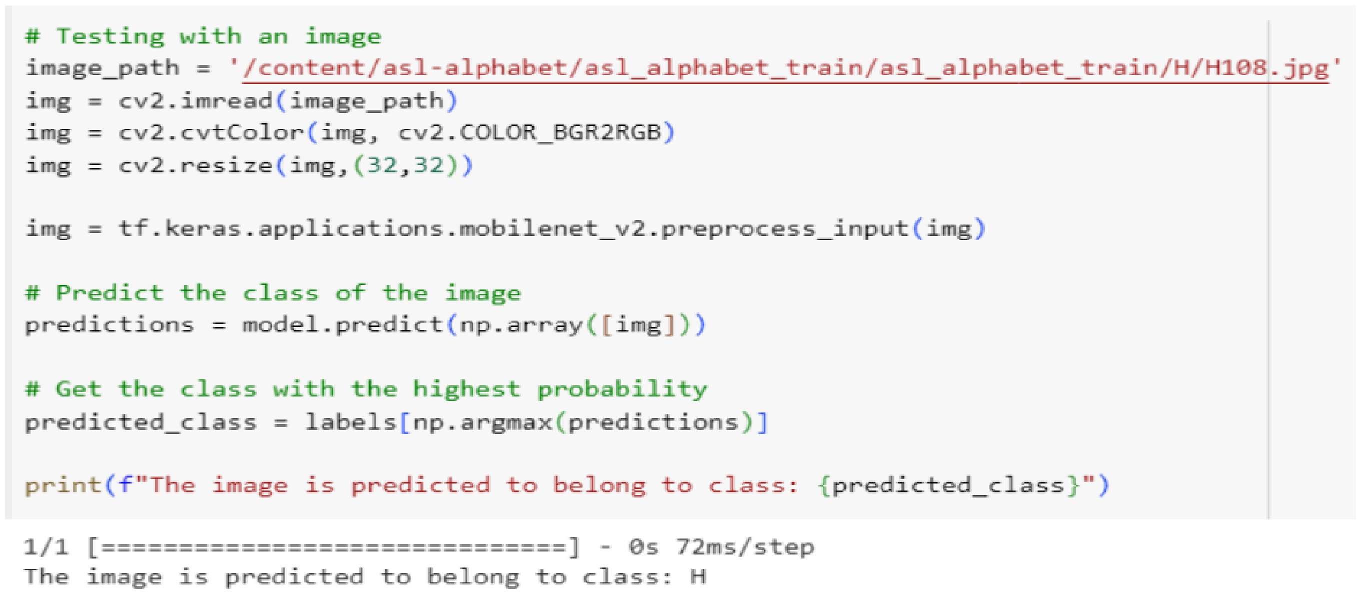
3.Classification Report:



```
136/136 [==============================] - 37s 271ms/step
              precision    recall  f1-score   support

           A       0.99      0.90      0.94       600
           B       0.96      0.94      0.95       600
           C       0.99      0.99      0.99       600
           D       0.99      0.97      0.98       600
           E       0.96      0.90      0.93       600
           F       1.00      0.96      0.98       600
           G       0.95      0.97      0.96       600
           H       0.98      0.94      0.96       600
           I       0.96      0.94      0.95       600
           J       0.96      0.98      0.97       600
           K       0.92      0.95      0.93       600
           L       0.99      0.99      0.99       600
           M       0.81      0.98      0.88       600
           N       0.94      0.87      0.91       600
           O       0.96      0.98      0.97       600
           P       0.98      0.98      0.98       600
           Q       0.97      0.99      0.98       600
           R       0.91      0.89      0.90       600
           S       0.84      0.94      0.89       600
           T       0.99      0.94      0.96       600
           U       0.92      0.90      0.91       600
           V       0.90      0.89      0.90       600
           W       0.97      0.95      0.96       600
           X       0.90      0.95      0.92       600
           Y       0.96      0.97      0.96       600
           Z       0.97      0.96      0.96       600
         del       0.98      0.97      0.98       600
     nothing       0.99      1.00      0.99       600
       space       0.98      0.98      0.98       600

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

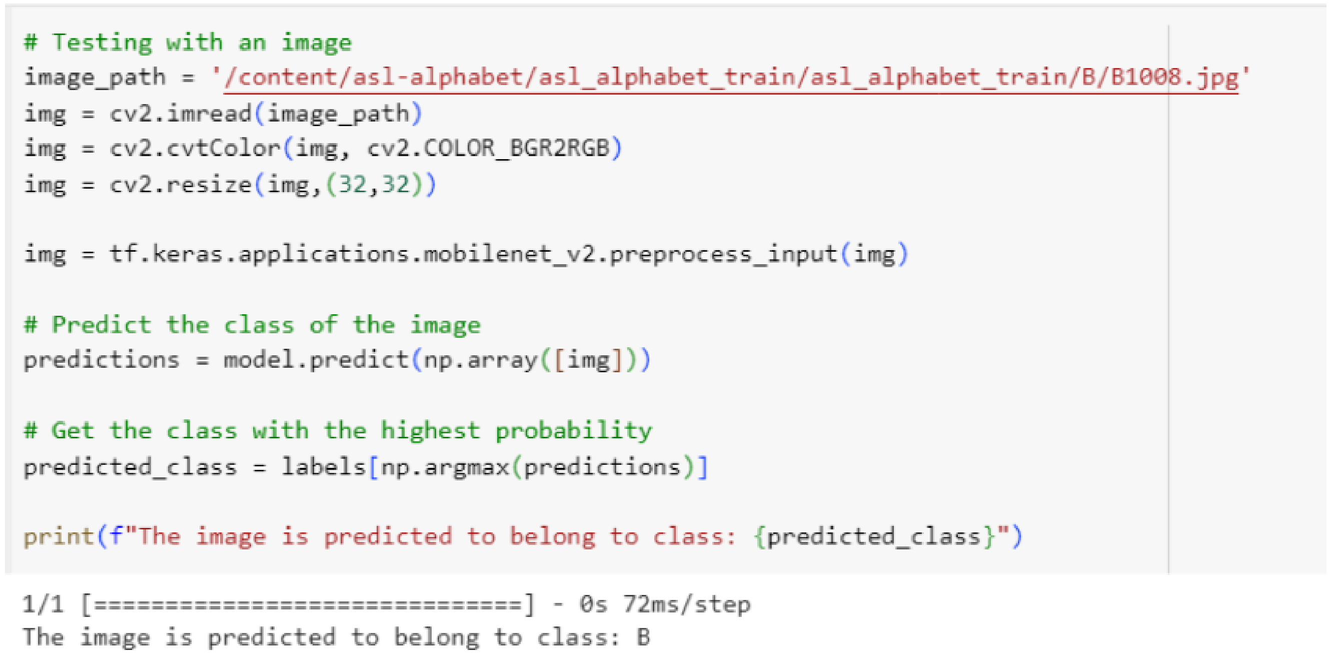
9. RESULTS

9.1 Output Screenshots



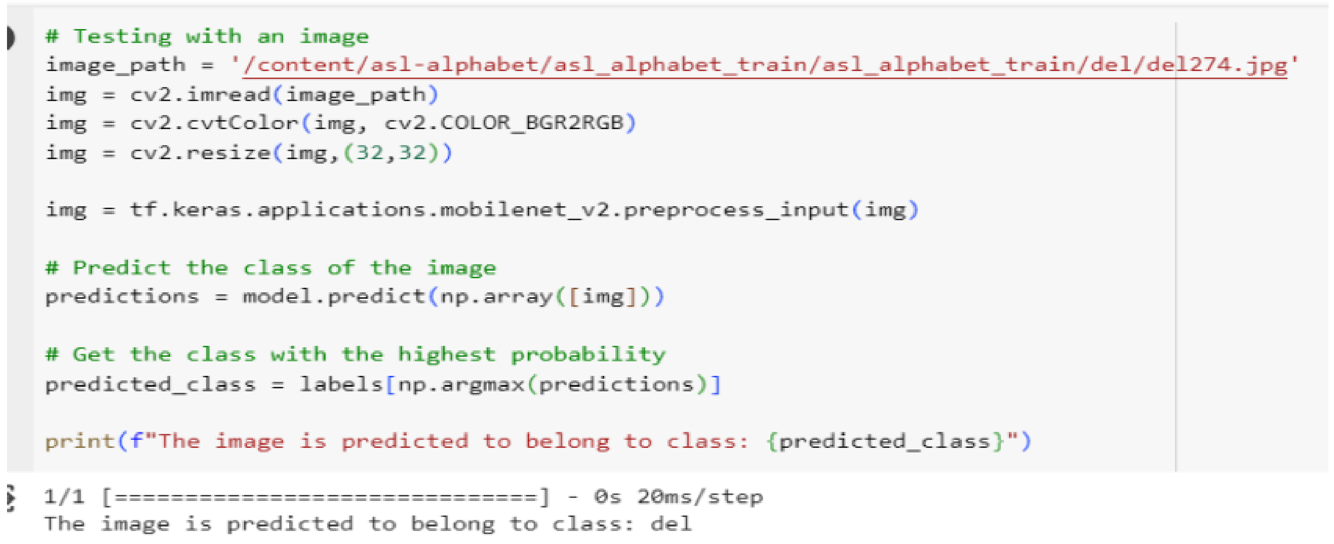
```python
# Testing with an image
image_path = '/content/asl-alphabet/asl_alphabet_train/asl_alphabet_train/H/H108.jpg'
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img,(32,32))

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
```



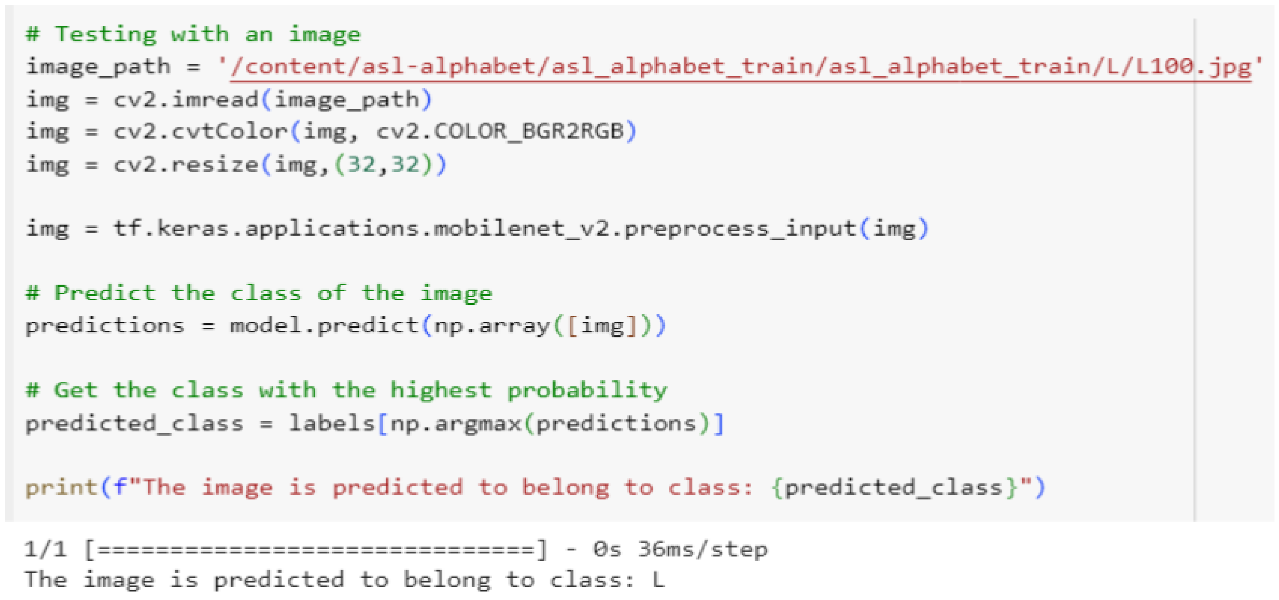
```
# Testing with an image
image_path = '/content/asl-alphabet/asl_alphabet_train/asl_alphabet_train/B/B1008.jpg'
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img,(32,32))

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
```



```
# 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:

1. **Accessibility Enhancement: It significantly improves accessibility for the Deaf and Hard of Hearing community by providing a tool to convert ASL fingerspelling into text, enabling better communication between those who use ASL and those who don't.**
2. **Communication Empowerment: Individuals within the Deaf community gain increased autonomy and empowerment in their interactions, as they can easily transcribe their messages into written text.**
3. **Technological Inclusivity: It promotes inclusivity by integrating ASL into technological advancements, bridging the gap between spoken and signed languages within the digital landscape.**
4. **Educational Support: ASL Alphabetic Image Recognition can aid in educational settings, assisting teachers and students in learning and practicing fingerspelling and enhancing the teaching of ASL.**

Disadvantages:

1. **Accuracy Challenges: ASL fingerspelling involves intricate hand movements, which can be challenging for image recognition systems to interpret accurately, leading to errors in transcription.**
2. **Variability in Gestures: Variations in signing styles and individual**

gestures can pose difficulties for recognition systems, as there's no standardized way of performing fingerspelling in ASL.

3. Complexity of Context: Understanding the context in which signs are used (such as slang or regional variations) can be challenging for the technology, potentially leading to misinterpretations.
4. Technological Limitations: Current technology might struggle with real-time recognition, causing delays in transcription and hindering fluid conversations.
5. Data Representation: The availability and quality of labeled datasets for training the recognition systems might be limited, affecting the system's ability to learn and improve accuracy.

11. CONCLUSION

ASL Alphabetic Image Recognition is a groundbreaking technology that aims to improve communication for the Deaf and hard-of-hearing. It uses machine learning and computer vision to decipher American Sign Language nuances and promotes linguistic diversity. This technology not only enhances the Deaf community's ability to express and connect, but also represents a paradigm shift in the field of technology. As it evolves, it sets a precedent for inclusive technology that adapts to diverse communication needs, inspiring a future where barriers dissolve and understanding transcends differences.

12. FUTURE SCOPE

Advancements in ASL Alphabetic Image Recognition (ASL) technology are expected to enhance accessibility, improve accuracy, and speed, and promote cultural integration.

This technology can be integrated into devices and applications, enabling on-the-go translation and communication for the Deaf community.

The technology will also focus on gesture recognition, capturing and interpreting complex signs and facial expressions.

It can also be used in educational tools to teach ASL to non-native speakers and support Deaf individuals in learning written language through fingerspelling.

Future developments will focus on enhancing accuracy and speed, collaborating with the Deaf community, and expanding the system's multilingual capabilities.

13. APPENDIX

Source Code

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>

    <style>

        body {

            font-family: Arial, sans-serif;

            background-color: #f0f0f0;

            margin: 0;

            padding: 0;

        }

        header {

            background-color: #333;

            color: white;

            text-align: center;

            padding: 20px 0;

        }

        .container {

            max-width: 600px;

            margin: 20px auto;

            padding: 20px;

            background-color: white;
            border-radius: 8px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        .info-section {
            margin-bottom: 20px;
        }
        #image-preview {
            margin-bottom: 20px;
            text-align: center;
        }
        #result {
            text-align: center;
            font-weight: bold;
        }
        #upload-form {
            text-align: center;
            margin-bottom: 20px;
        }
        #upload-label {
            padding: 10px 20px;
            background-color: #4285f4;
            color: white;
            border-radius: 4px;
            cursor: pointer;
            transition: background-color 0.3s ease;
        }
        #upload-label:hover {
            background-color: #3367d6;
        }
        #predict-button {
            padding: 10px 20px;
```

```
            background-color: #34b7f1;
            color: white;
            border: none;
            border-radius: 4px;
            cursor: pointer;
            transition: background-color 0.3s ease;
        }
        #predict-button:hover {
            background-color: #1e90ff;
        }
    </style>
</head>
<body>
    <header>
        <div class="header-content">
            <h1>American Sign Language Recognition</h1>
            <p></p>
        </div>
    </header>

    <div class="container">
        <div class="info-section">
            <p>ASL stands as the principal language utilized by North American
individuals who are deaf. It operates as a visual language, relying on a fusion
of hand motions, facial cues, and bodily gestures to communicate profound
meanings.
            </p>
        </div>

        <!-- Upload Image Form -->
        <form id="upload-form" action="/predict" method="post"
enctype="multipart/form-data">
```

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

            <label for="file-input" id="upload-label">Choose Image</label>

        </form>


        <!-- Display Image -->

        <div id="image-preview"></div>


        <!-- Predict Button -->

        <button id="predict-button" onclick="predict(event)">Predict</button>


        <!-- Display Prediction Result -->

        <p id="result"></p>

    </div>


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

</body>

</html>
```

Java Script:

```javascript
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 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);
    fetch('/predict', {
        method: 'POST',
        body: formData
    })
    .then(response => response.json())
    .then(data => {
        resultElement.innerText = 'Prediction: ' + data.prediction;
    })
    .catch(error => console.error('Error:', error));
}
```

App.py:

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

```python
app = Flask(__name__)

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

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

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

        file = request.files['file']

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

        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)
        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 = np.expand_dims(img_array, axis=0)  # Add batch dimension
        # Make prediction

        prediction = model.predict(img_array)


        predicted_class = labels[np.argmax(prediction)]


        text = "Your image represents "+predicted_class

        return jsonify({'prediction': text})



if __name__ == '__main__':

    app.run(debug=False, threaded = False)
```

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)]
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}")
```

GitHub & Project Demo Link

https://github.com/smartinternz02/SI-GuidedProject-615313-1699449676/upload

Project demo link:

https://cpf-temp-repo-an1-prod.s3.ap-northeast-1.amazonaws.com/d68d1eb4-2da2-4cec-924b-59082f4ab7b4?X-Amz-Security-Token=FwoGZXIvYXdzEOf%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaDOvNWeq%2BDPFv1oU1vSLcAULr%2BSVhGOKkR%2FxsfIXB6nvtD7RawXu%2FGpv99YJj%2BmxkBLi8iHtKA991KZ7EBhEIb6nJ21dJOcR1tGCOvF3pvrziIFy8Cc7%2FyofZXIsa%2FFLix4ocNWEkBQ4N8sD4JkI1DHyJdmnOGwUTiO7eGjO5SfOUN35s74f7bvmchWr1ywJuoHAMQjM68ADWSiqzJ84PRVRqkSn9uAFAcPfzL%2BvQxrPPGZjnEbFItSqQqChP3kA5wL6GwdQIdUNO9MrpIyePobI9D%2FdFpqV%2BtI%2Bf7dbmPxhWfxC6Ydqk7BVR65Eom5D4qgYyLbKpCp5%2B7MBHWB5susmCVOMIQZnUGn48RIRI11B5kIkmeIk6UmpvRpYWhSAigw%3D%

3D&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20231122T140625Z&X-Amz-SignedHeaders=host&X-Amz-Expires=86400&X-Amz-Credential=ASIAU3E5TYZ25EDLLM60%2F20231122%2Fap-northeast-1%2Fs3%2Faws4_request&X-Amz-Signature=317794cbd1bbba97c99e9292342903af0793e57df02bff47a6d45699922959cd