A Project Report

On

American Sign Language Prediction System using ML

For an Internship Program of 6 Months

Submitted To:

UNIFIED MENTOR

YOUR SKILL, SUCCESS & JOURNEY

Submitted By:

Karmendra Bahadur Srivastava

Machine Learning Intern (6 Months)

UMID02042527293

Made In:
30th of August, 2025

Intern Programme Duration:
APRIL to OCTOBER 2025

Candidate's Declaration

I hereby declare that the project titled "American Sign Language Prediction System using ML" submitted is an original work carried out by me.

I further declare that the work presented in this project does not violate or infringe on the rights of any third party and is in complete compliance with academic and ethical standards. All sources of information and data have been properly cited and acknowledged.

Furthermore, I declare that this project, including all its contents, structure, code-base, and data processing logic, is my intellectual property and is protected under applicable copyright laws.

This project has been created strictly for academic, educational, and research purposes only. Any reproduction, reuse, distribution, or modification of this project — either partially or fully — is only permitted for non-commercial, fair use, with proper acknowledgment of the original creator.

Karmendra Bahadur Srivastva
karmendra5902@gmail.com
UMID02042527293
30th of August, 2025

Acknowledgement

I would like to express my sincere gratitude to all those who supported and guided me throughout the development of this project titled "ASL Prediction System using ML".

I am especially thankful to my mentor/supervisor “Unified Mentor” for providing me with the database for training and the instructions throughout the duration of the project.

Furthermore, I am grateful to the open-source developer community and contributors for the tools, libraries, and resources that made this project possible.

Also the Various Sources for Compatibility, Usability and Portability with different systems that made the project as polished and possible to be worked on the future.

Lastly, I acknowledge the support of my institution in providing the education and experience to work on and present this project report.

Karmendra Bahadur Srivastava

30th, August 2025

Abstract

Communication is a fundamental human need, yet individuals with hearing and speech impairments often face barriers when interacting with others. American Sign Language (ASL) is one of the most widely used mediums of communication in such communities, but it is not universally understood. To bridge this gap, my project presents a system capable of automatically detecting ASL hand gestures and mapping them to their corresponding English letters and control commands (SPACE, DELETE, NOTHING).

The solution leverages a deep learning model trained on a large-scale ASL data-set consisting of 29 classes, including the 26 letters of the alphabet and three additional symbols. Using MobileNetV2 as the backbone architecture, the system achieves efficient training and accurate classification of static hand gestures. Beyond the model, I have designed an interactive Streamlit-based application that enables users to interact with the system through multiple modes—uploading images, using live camera detection, constructing words through sequential gestures, and exploring sample references with an integrated quiz game.

To enhance usability, the application maintains a detailed history of user interactions, supports export in multiple formats, and provides an intuitive interface suitable for both beginners and advanced users. The project thus demonstrates how deep learning, combined with user-centered design, can contribute toward accessible, real-time sign language interpretation tools that promote inclusivity and communication in everyday life.

List of Table

List of Figures

List of Abbreviation

S.No	**Word**	**Abbreviation**
6.1	American Sign Language	ASL
6.2	Portable Document Format	PDF
6.3	Streamlit	ST
6.4	Comma-Seperated Values	CSV
6.5	JavaScript Object Notation	JSON
6.6	Hierarchical Data Format V5	HDF5
6.7	Batch files	bat
6.8	Jupyter Notebook	Jupyter
6.9	Joint Photographic Experts Group	JPG or JPEG
6.10	American Standard Code of Information Interchange	ASCII
6.11	Exploratory Data Analysis	EDA
6.12	Receiver Operating Characteristic	ROC
6.13	Principal Component Analysis	PCA
6.14	t-Distributed Stochastic Neighbor Embed	t-SNE
6.15	Convolutional Neural Networks	CNNs
6.16	Deep Learning	DL
6.17	TensorFlow	Tf
6.18	Pickle	pkl
6.19	Graphical User Interface	UI
6.20	User Experience	UX
6.21	Wev Real-Time Communication	WebRTC

Table of Content

Introduction

7.1 Problem Statement

In Sign languages such as American Sign Language (ASL) serve as a vital means of communication for individuals who are deaf or hard of hearing. However, the lack of widespread knowledge of ASL among the general population often creates barriers to effective communication in education, workplaces, healthcare, and daily interactions. Human interpreters provide an essential bridge, but they are not always available, practical, or affordable.

This situation highlights the need for intelligent systems that can recognize and interpret ASL gestures into textual or spoken output in real time. Traditional approaches to gesture recognition are limited by handcrafted feature extraction and inconsistent accuracy across diverse hand shapes, lighting conditions, and user variations. Advances in deep learning and computer vision, however, offer new opportunities to build robust, scalable systems capable of understanding ASL with high accuracy.

Dataset:

The dataset used in this project contains 29 classes, covering the 26 English alphabets (A–Z) and three functional symbols (SPACE, DELETE, NOTHING). It is structured into separate training and testing sets, where the training data includes approximately 3,000 images per class, representing diverse hand orientations and backgrounds, while the test set provides one representative image per class for evaluation. This dataset supports both model development and system-level testing.

By leveraging this dataset and applying state-of-the-art deep learning models, our project aims to bridge the communication gap through a real-time ASL gesture detection system integrated into a user-friendly application.

7.2 Objective of the Project

The primary objective of this project is to develop an intelligent ASL gesture detection system that recognizes static signs and translates them into their corresponding alphabetic characters or control commands. Beyond detection, the project emphasizes interactivity, accessibility, and educational value through a carefully designed interface.

a) Accurate Gesture Recognition – Train and deploy a MobileNetV2-based deep learning model to achieve high classification accuracy across all 29 ASL classes.

b) Interactive User Interface – Build a modular Streamlit application with intuitive navigation, allowing users to upload images, use live camera input, and construct words interactively.

c) Word Construction and Validation – Enable sequential gesture inputs to form words, with dictionary-based validation for meaningful interpretation.

d) History and Export Features – Provide a structured history log of predictions, with options to download results in multiple formats (.pdf, .csv, .json).

e) Learning and Engagement – Offer reference gestures for practice and a quiz-based game mode to enhance learning and user engagement.

f) Extendable Design – Maintain modular code and configuration to support potential future features such as speech synthesis, advanced sentence construction, or real-time translation in video streams.

7.3 Overview of the Operations

7.3.1 Overview of the System

The project of American Sign Language Detection System integrates deep learning, a web-based interface, and interactive utilities into a cohesive platform.

The system is composed of several major components:

a) Backend Model – A MobileNetV2-based deep learning model trained on the ASL dataset. The model is saved in .h5 format with accompanying class labels in JSON for efficient inference.

b) Frontend Application (Streamlit) – The application provides multiple modes: Upload Prediction, Live Camera Detection, Word Maker, History, and Sample Gestures. Each module is designed with accessibility and usability in mind.

c) History Management – All predictions are logged into a structured JSON file, with options for export and visualization in tabular format.

d) Word Maker Module – Sequential gesture captures form words, which are validated against a dictionary to distinguish real words from random combinations.

e) Sample Gestures & Quiz Game – A learning aid that provides reference images and an interactive quiz mode for users to test their knowledge of ASL symbols.

f) Utility Scripts & Automation – Batch files for environment setup (install_modules.bat, run.bat), configuration files, and modular Python scripts streamline installation, execution, and further development.

7.3.2 Relevance

The proposed system holds significance across multiple dimensions. For individuals unfamiliar with ASL, it offers an accessible medium to learn and interact with sign language. For members of the deaf and hard-of-hearing community, it represents a step toward bridging communication gaps in everyday contexts. Educators and learners can also benefit from its interactive modules and gamified quiz feature.

From a technical perspective, the project demonstrates how deep learning models can be deployed within lightweight, interactive applications to solve real-world accessibility challenges. Its modular and extendable design ensures that the system can evolve to incorporate additional languages, sentence-level recognition, or speech synthesis in the future.

Thus, this project not only provides a working ASL detection tool but also showcases the potential of AI-driven accessibility solutions that blend accuracy, usability, and inclusivity.

Data Collection and Preprocessing

8.1 Data Source and Formatting

The dataset used in this project is derived from the United Machine Learning Repository, The American Sign Language Alphabet Dataset. designed specifically for training and evaluating machine learning models on static American Sign Language gestures. It contains 29 classes, representing the 26 English alphabets (A–Z) and three additional symbols: SPACE, DELETE, and NOTHING.

It contains around 86,000 images with different attributes in each of the image, Resulting in a large and Balanced Dataset suitable for the Machine Learning Work with the help of Deep Learning. Out of the estimated 86,000 images it is classified into 29 Categories of Different Characters with an approximately 3,000 images per class which helped the system to clearly understand the Posture and shape of the Hand when detection.

The Testing Set contained a single representative image of each class, enabling quick and easy evaluation with verification.

The images are provided in .jpg format, organized in class-specific folders. The hierarchical structure makes it straightforward to load and label images using data loaders.

Sample of Dataset Categories:

Table 1: Details about the Dataset

Characters	Count	Size
A - Z (ASCII)	26	77,000
SPACE	1	3,000
DELETE	1	3,000
NOTHING	1	3,000

The dataset was well-structured, with well sturtured folder and file names, As for Test Entires they are A_test.jpg, B_test.jpg and for Train Set: A/A1.jpg,A2.jpg, B/B1.jpg, B2.jpg and so on.

The Images were in various States to train the Model with Efficiency and Easily Recognized Patterns when we Deploy the Deep Learning Model. Here are the Sample of some Images that can clearly show how the samples of dataset are truly different to give the model as easier time to recognized any in the future.

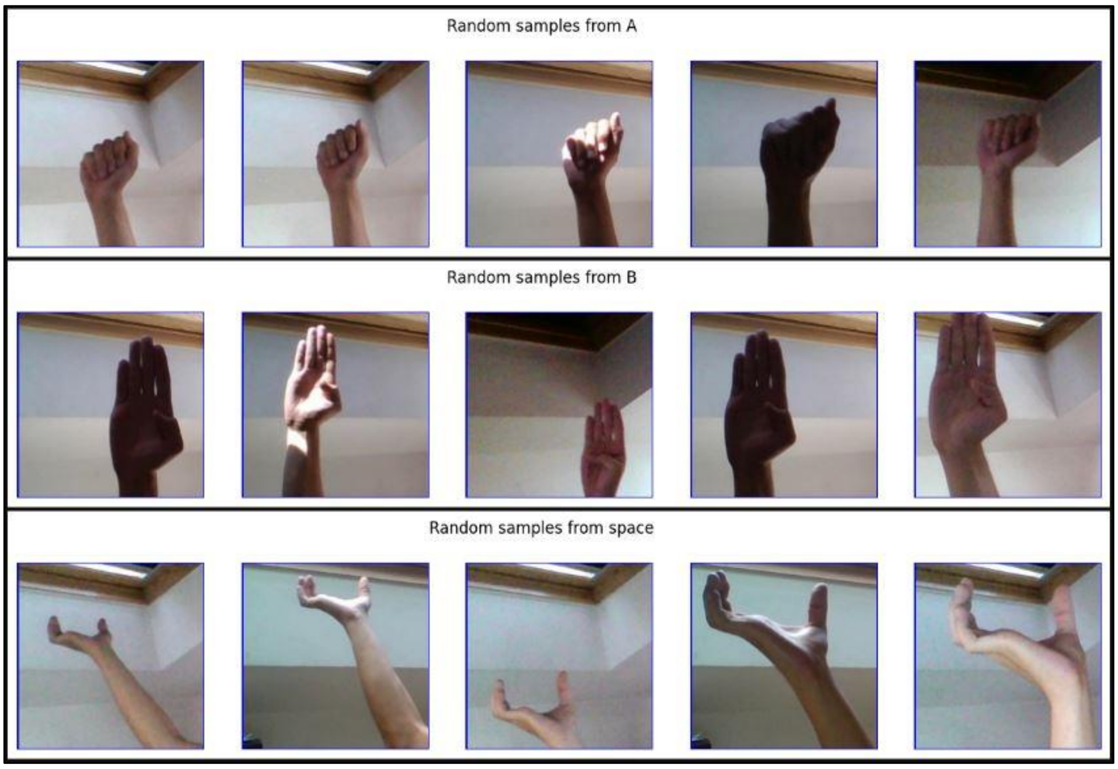


Figure 1: Sample Hand Gestures in Various Conditions

The dataset captures hand gestures can be seen by the Random Samples consisting under varying conditions, such as changes in hand orientation, background clutter, and lighting variations, making it robust for training real-world recognition models.

8.2 Data Exploration and Insights

Although before model training, exploratory data analysis (EDA) was conducted to gain a deeper understanding of the dataset's structure, distribution, and variability. This step was essential to validate the dataset's quality and to inform preprocessing and modeling decisions. Jupyter Notebook was used for visualization and statistical inspection. The key findings are summarized below:

8.2.1 Uniform Class Distribution

Each gesture class contained approximately 3,000 samples, ensuring a balanced dataset. This eliminates the risk of model bias toward more frequent classes and reduces the need for resampling techniques such as oversampling or class weighting. The uniform distribution also supports fair evaluation metrics across all classes.

8.2.2 Image Dimensions and Consistency

All images were stored at a consistent resolution, which simplified preprocessing. To further standardize the data, images were resized and normalized to a common scale suitable for the neural network input. This consistency ensured that model performance would not be influenced by variations in image size or pixel intensity ranges.

8.2.3 Variation in Appearance

The dataset included diverse examples of each gesture, with variations in hand shape, orientation, lighting conditions, and user characteristics. This diversity is crucial for enhancing the model's generalization capability, enabling it to perform well on unseen users and environments. Additionally, exploratory visualization highlighted that while gestures were visually distinct, some had subtle similarities, which could pose challenges during classification and require robust feature extraction.

8.2.4 Noise and Data Quality

A small number of samples exhibited slight blurring or background variation. Although not extensive, these variations mimic real-world conditions and may actually improve robustness by preventing overfitting to overly clean data. Outliers were rare and deemed not impactful enough to require removal.

8.2.5 Correlation Across Features

By visualizing pixel intensity distributions and feature embeddings (via PCA and t-SNE), it was observed that gesture classes formed distinguishable clusters, though with some overlap in visually similar gestures. This indicated that while the dataset is suitable for supervised learning, high-capacity models such as deep CNNs are required to capture fine-grained differences.

8.2.6 Suitability for Deep Learning

Overall, the exploratory analysis confirmed that the dataset is well-structured, balanced, and diverse. The presence of sufficient intra-class variability and inter-class separability justifies the use of deep learning models. Insights gained from this analysis guided preprocessing decisions such as normalization, augmentation, and input resizing, which were critical for model performance.

8.3 Preprocessing Steps

The dataset required preprocessing to transform raw .jpg images inputs into machine-learning-friendly formats. The original data was relatively clean, but necessary steps were taken to encode them in a certain way that functions with the program.

8.3.1 Image Resizing and Normalization

To ensure compatibility with deep learning architectures, all images were resized to 160 × 160 pixels and normalized to a [-1, 1] range. Normalization ensured consistent pixel intensity distribution, improving convergence during training.

These encoded versions were saved and late loaded in the deployed application using joblib library.

8.3.2 Label Encoding

Each folder name (e.g., "A", "B", "SPACE") was mapped to a numerical label from 0–28. A JSON file (class_names.json) was generated to store the mapping, ensuring consistency between training, inference, and user interface display.

8.3.3 Data Augmentation

Since real-world usage involves variations in lighting, orientation, and scale, augmentation techniques were applied during training to increase dataset diversity. Augmentation strategies included:

- Random rotations (±20°)
- Horizontal flips
- Zooming and shifting
- Brightness and contrast adjustments

This step improved the robustness of the model against unseen data.

8.3.4 Model Readiness

The preprocessing workflow ensured that the machine learning model was trained on consistent, high-quality data, which enabled accurate predictions in the live system. The trained model and scaler were both exported using joblib and loaded at runtime using a lightweight script.

a) Normalize features

b) Encoding sizes

c) Splitting model into train/validation/test model.

d) Validating prediction on test set

e) Exporting model

f) Integration with Front-end by creating the model using joblib library.

This structured approach ensured that the model received optimal data quality allowed performance monitoring and reduced over-fitting risks ,improving both accuracy and interpret-ability during deployment in the final Streamlit application.

8.3.5 Why Preprocessing

Without preprocessing, inconsistencies in image size, orientation, and intensity could significantly reduce model accuracy. The preprocessing pipeline ensured:

- **Consistency:** Uniform input size and normalized values.
- **Robustness:** Augmentation increased resistance to noise and real-world variations.
- **Scalability:** The modular data loader allows seamless integration of future datasets or extended ASL vocabularies.

Thus, preprocessing was a critical foundation that enabled the successful deployment of a reliable and interactive ASL detection system.

Model Evaluation and Training

9.1 Technology and Model Used

This project uses deep learning with computer vision to recognize American Sign Language (ASL) hand gestures. The primary objective was to build a reliable classification system capable of distinguishing between 29 gesture classes, including letters A–Z (except J and Z, which require motion) and three special tokens (space, delete, nothing).
The system integrates several modern technologies across machine learning, visualization, and deployment:
a) Python: The primary programming language that comforts me for both data science and web development in this project due to its readability and strong ecosystem.
b) Tensorflow/Keras: They are core framework for deep learning model development.
c) Jupyter Notebook: Used for exploratory data analysis (EDA), data preprocessing, model experimentation, and training. Its interactive environment provides a readable interface enabled for quick iteration and visual insights.
d) MobileNetV2: A lightweight yet powerful convolutional neural network, pre-trained on ImageNet, chosen for its balance of accuracy and efficiency.
e) Scikit-learn: Provided essential utilities for data preprocessing, model training, evaluation metrics, and model serialization.
f) Custom Classification Head: Added layers including Global Average Pooling, Dropout and Dense (Softmax output layer with 29 nodes).
g) Joblib: Utilized to serialize the trained model, allowing the web app to load predictions without retraining.
h) Streamlit: The framework used to create an interactive web interface for non-technical users to engage with the model. It enables real-time predictions, visual analytics, and theme customization.

i) **Matplotlib & Seaborn:** Used for generating visual insights into feature importance, dataset characteristics, and model performance.

This combination ensures that the model is both highly accurate and deployable on resource-constrained environments like web apps or edge devices.

9.2 Dataset and Class Distribution

Given the data-set used in this study contains tens of thousands of labeled images of American Sign Language (ASL) gestures, spanning 29 distinct classes. Each class corresponds to a unique hand sign, covering the 26 letters of the English alphabet along with three additional signs representing "space," "delete," and "nothing." This diversity ensures that the model can handle not only alphabetic gestures but also essential interaction commands required for practical ASL recognition systems.

9.2.1 Class Frequencies

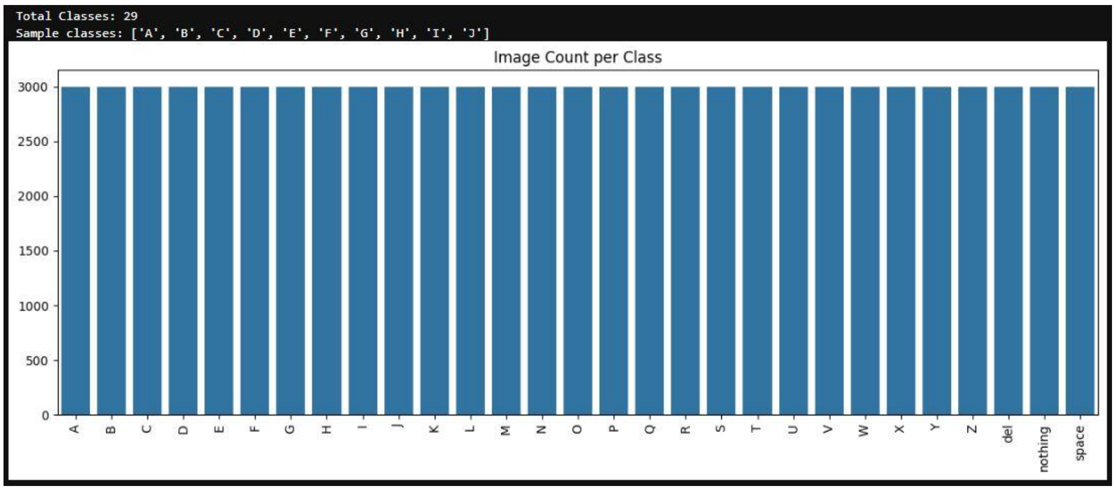
To better understand the dataset, a bar chart was generated to illustrate the frequency of samples across all gesture categories. The visualization highlighted that, while the dataset is generally well-balanced and equally Distrubuted among the Categories seen in Table1, Each and every one of the Class has around ~3,000 data images helped in the training of each and every model Character that will be helpful in the integration of my program.

9.2.2 Data Augmentation for Balance and Robustness

To mitigate the effects of class disparity and to enhance generalization, several data augmentation techniques were applied, including random rotations, horizontal flips, zooming, and shifting. These transformations served two purposes:

a) **Balancing Representation:** Some classes were conflicting with each other and were expanded to get accurate results, narrowing the gap between the similar kind of Hand Gestures.

b) **Improving Generalization**: Augmented variations prevented the model from memorizing fixed hand positions and backgrounds, enabling it to learn robust features that generalize well to unseen data.



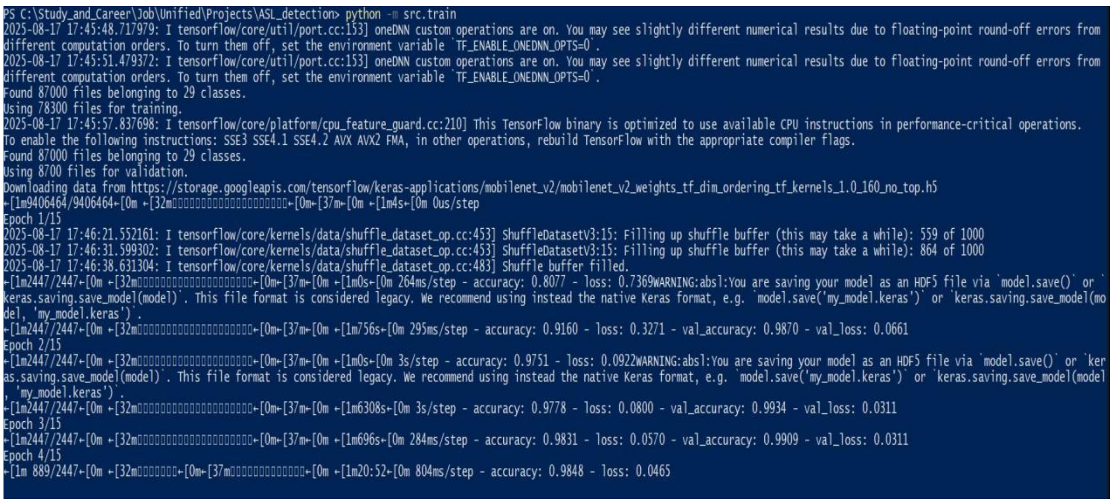
```
Total Classes: 29
Sample classes: ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
```

Figure 2: Classification of all the Characters

This preprocessing step proved critical, as it reduced overfitting and promoted fairness across all gesture categories, ultimately leading to a more accurate and reliable ASL recognition system.

9.3 Model Training Process

The training of the ASL recognition model followed a two-phase transfer learning strategy, leveraging MobileNetV2 as the feature extractor and gradually adapting it to the target dataset. This approach balanced computational efficiency with performance by first learning task-specific representations at the classification layer, then fine-tuning deeper layers of the backbone.



```
PS C:\Study_and_Career\Job\Unified\Projects\ASL_detection> python -m src.train
2025-08-17 17:45:48.717979: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from
different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-08-17 17:45:51.479372: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from
different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
Found 87000 files belonging to 29 classes.
Using 78300 files for training.
2025-08-17 17:45:57.837698: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Found 87000 files belonging to 29 classes.
Using 8700 files for validation.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_160_no_top.h5
←[1m9406464/9406464←[0m ←[32m□□□□□□□□□□□□□□□□□□□□←[0m←[37m←[0m ←[1m4s←[0m 0us/step
Epoch 1/15
2025-08-17 17:46:21.552161: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:453] ShuffleDatasetV3:15: Filling up shuffle buffer (this may take a while): 559 of 1000
2025-08-17 17:46:31.599302: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:453] ShuffleDatasetV3:15: Filling up shuffle buffer (this may take a while): 864 of 1000
2025-08-17 17:46:38.631304: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:483] Shuffle buffer filled.
←[1m2447/2447←[0m ←[32m□□□□□□□□□□□□□□□□□□□□←[0m←[37m←[0m ←[1m0s←[0m 264ms/step - accuracy: 0.8077 - loss: 0.7369WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `
keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(mo
del, 'my_model.keras')`.
←[1m2447/2447←[0m ←[32m□□□□□□□□□□□□□□□□□□□□←[0m←[37m←[0m ←[1m756s←[0m 295ms/step - accuracy: 0.9160 - loss: 0.3271 - val_accuracy: 0.9870 - val_loss: 0.0661
Epoch 2/15
←[1m2447/2447←[0m ←[32m□□□□□□□□□□□□□□□□□□□□←[0m←[37m←[0m ←[1m0s←[0m 3s/step - accuracy: 0.9751 - loss: 0.0922WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `ker
as.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model
, 'my_model.keras')`.
←[1m2447/2447←[0m ←[32m□□□□□□□□□□□□□□□□□□□□←[0m←[37m←[0m ←[1m6308s←[0m 3s/step - accuracy: 0.9778 - loss: 0.0800 - val_accuracy: 0.9934 - val_loss: 0.0311
Epoch 3/15
←[1m2447/2447←[0m ←[32m□□□□□□□□□□□□□□□□□□□□←[0m←[37m←[0m ←[1m696s←[0m 284ms/step - accuracy: 0.9831 - loss: 0.0570 - val_accuracy: 0.9909 - val_loss: 0.0311
Epoch 4/15
←[1m 889/2447←[0m ←[32m□□□□□□□←[0m←[37m□□□□□□□□□□□□□←[0m ←[1m20:52←[0m 804ms/step - accuracy: 0.9848 - loss: 0.0465
```

Figure 3: Insights of Training The Model

Phase 1: Feature Extraction: In the initial stage, the MobileNetV2 base model—pretrained on the ImageNet dataset—was used as a frozen feature extractor. All convolutional layers were locked to preserve the generalized visual features (edges, textures, shapes) learned during large-scale pretraining. Only the custom classification head (a fully connected layer stack appended to the base) was trainable during this phase.

a) Optimizer: Adam, with a learning rate of 0.001, provided stable convergence with adaptive learning rate adjustments for each parameter.
b) Loss Function: Sparse Categorical Crossentropy was chosen since the dataset contained integer-encoded class labels rather than one-hot vectors, reducing memory usage while maintaining efficiency.
c) EarlyStopping: Implemented with a patience of 4 epochs, this callback monitored validation loss and halted training if performance stopped improving, preventing overfitting and unnecessary computations.

This phase ensured that the classification head learned to map high-level features from MobileNetV2 into the 29 ASL categories without disrupting the pretrained weights.

Phase 2: Fine-Tuning: Once the classification head achieved stable performance, the second phase involved fine-tuning the deeper layers of MobileNetV2 to adapt the feature representations more closely to ASL hand gestures. Specifically, all layers after index 100 in the base network were unfrozen, while earlier layers remained frozen to preserve low-level, general-purpose feature extraction.

a) Reduced Learning Rate: A smaller learning rate of 0.0001 was used during this phase. This careful adjustment ensured that fine-tuning updated weights gradually, avoiding catastrophic forgetting of pretrained knowledge.
b) End-to-End Training: The model was trained across the full architecture, enabling both the base network and classification head to co-adapt to the nuances of hand shapes, orientations, and subtle differences in ASL signs.

This phase significantly boosted performance, as the model not only recognized broad visual patterns but also refined its sensitivity to the domain-specific intricacies of hand gestures.

Model Preservation for Deployment: Upon convergence, the best-performing model was saved in the HDF5 format as asl_mobilenetv2.h5. This format ensured compatibility across platforms and facilitated deployment in real-time applications, such as ASL recognition systems integrated into web or mobile interfaces. By saving the final trained model, reproducibility was guaranteed, and the model could be reloaded without retraining, enabling efficient experimentation and deployment.

9.4 Evaluation Metrics

To assess the reliability and generalization, the model was evaluated using multiple metrics. We will find out more about the insight data and such more delighted results.

9.4.1 Confusion Matrix

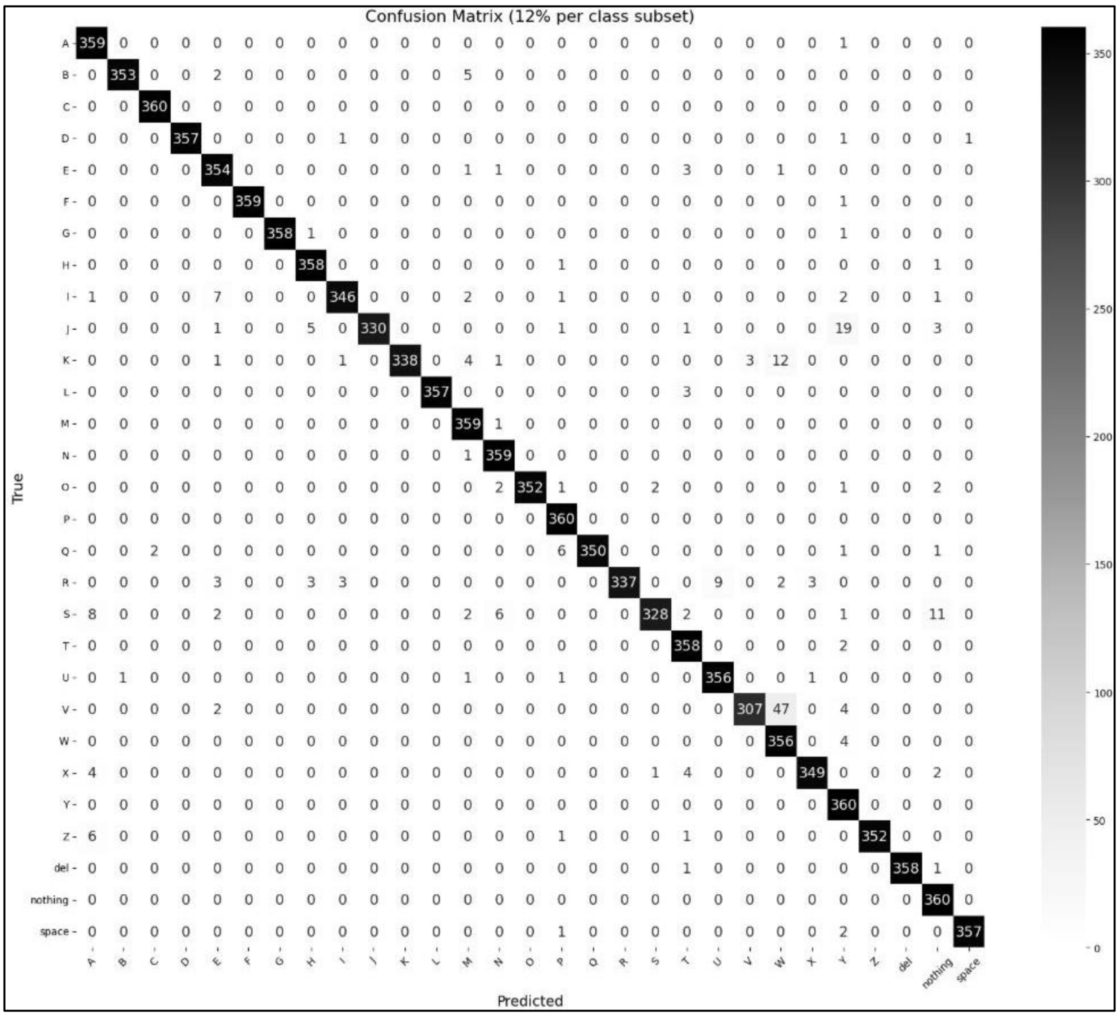
The confusion matrix reveled high per-class accuracy, It is used to represents the various relations and occurance of similarilty from the dataset for the other categories as well but most importantly highlighted specific misclassifications, such as:

a) V vs W: Frequently confused due to similar finger configurations.

b) J vs Y: Occasional errors because of subtle placement differences.

c) K vs W: The Shape is Similar to the V vs W situation that may confuse a little bit.

Because of the Size of the Data-set, I picked the 12% of the subset to accuractly represents the missclassifications that the full of all the Data-set should be showing.



	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
A	359	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
B	0	353	0	0	2	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	357	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
E	0	0	0	0	354	0	0	0	0	0	0	0	1	1	0	0	0	0	0	3	0	0	1	0	0	0	0	0	0
F	0	0	0	0	0	359	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	358	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
H	0	0	0	0	0	0	0	358	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
I	1	0	0	0	7	0	0	0	346	0	0	0	2	0	0	1	0	0	0	0	0	0	0	0	2	0	0	1	0
J	0	0	0	0	1	0	0	5	0	330	0	0	0	0	0	1	0	0	0	1	0	0	0	0	19	0	0	3	0
K	0	0	0	0	1	0	0	0	1	0	338	0	4	1	0	0	0	0	0	0	0	3	12	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	357	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	359	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	1	359	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	0	2	352	1	0	0	2	0	0	0	0	0	1	0	0	2	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0	0	0	0	0	0	0	0	0	0	0	0	0
Q	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	6	350	0	0	0	0	0	0	0	1	0	0	1	0
R	0	0	0	0	3	0	0	3	3	0	0	0	0	0	0	0	0	337	0	0	9	0	2	3	0	0	0	0	0
S	8	0	0	0	2	0	0	0	0	0	0	0	2	6	0	0	0	0	328	2	0	0	0	0	1	0	0	11	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	358	0	0	0	0	2	0	0	0	0
U	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	356	0	0	1	0	0	0	0	0
V	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	307	47	0	4	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	356	0	4	0	0	0	0
X	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	0	0	0	349	0	0	0	2	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0	0	0	0
Z	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	352	0	0	0
del	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	358	1	0
nothing	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	360	0
space	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	2	0	0	0	357

Figure 4: Confusion Matrix of 12% Distrubuted Dataset

9.4.2 Misclassification Set

To better understand the model's weaknesses, a misclassification set was curated with a total of 263 in the 12% of the dataset. This subset consisted of images where the predicted label differed from the ground truth, allowing for a direct inspection of the errors made by the model. Each misclassified example was annotated with both the true label and the predicted label, making it easier to identify recurring patterns of confusion.

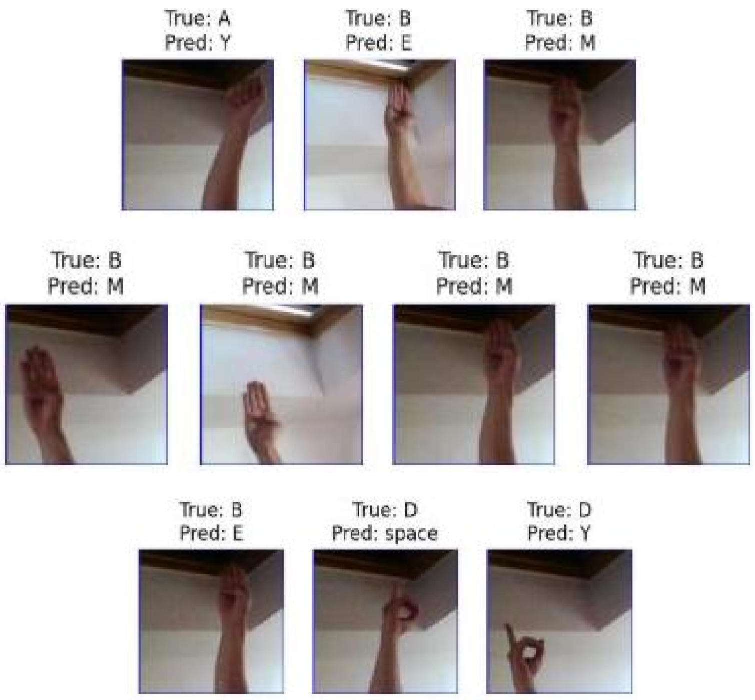


Figure 5: True vs Predicted Values of a sample of the data-set

9.4.3 Class Distribution and ROC curves

Unlike binary classification problems, where ROC curves provide valuable insight into model performance, this project involves multi-class classification across various distinct result types. In such cases, confusion matrices and per-class accuracy provide a clearer and more interpretable measure of the model's strengths and weaknesses.

A detailed confusion matrix highlights how well the model differentiates between The different price zones that might be have the same predicted probability, which are often confused due to overlapping in different arrangments of inputs.

Additionally, the class distribution analysis revealed a relatively balanced dataset, minimizing bias toward dominant classes.

Key Observations:

a) The model performed particularly well on the accuracy of the model with around ~98%.

b) Minor miss-classification occurred between the concurrent types.

c) The Results are Almost tending to 1 with a few expectations.

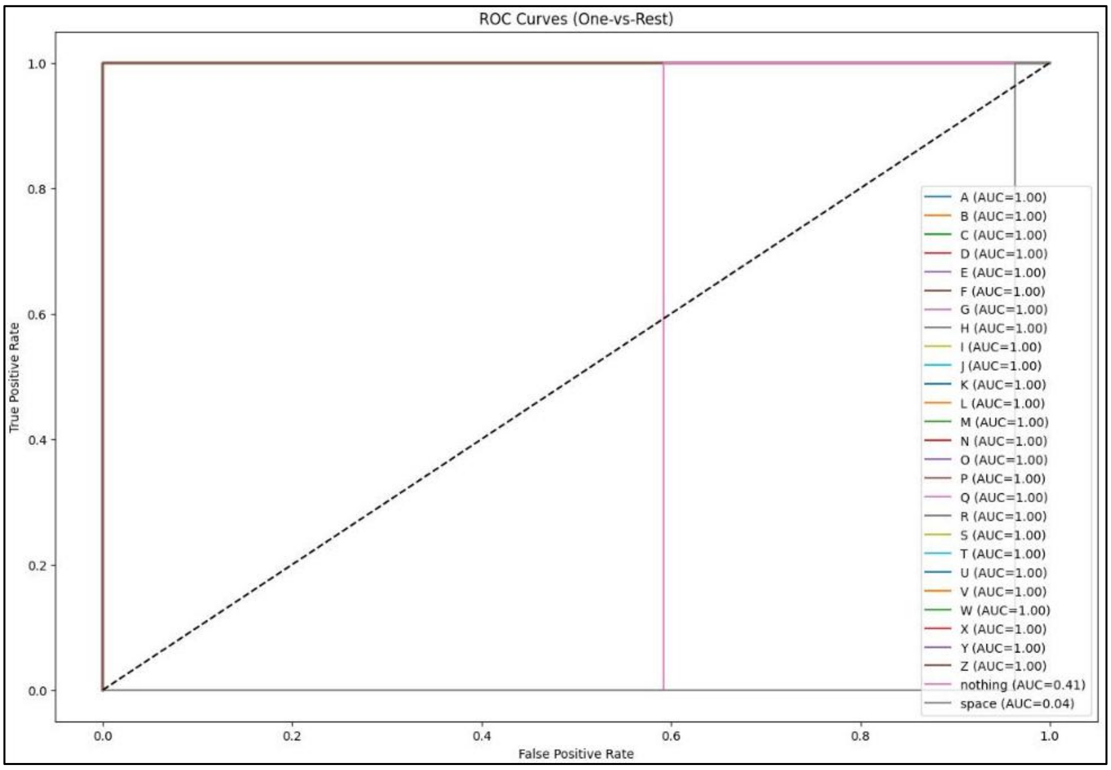


Figure 6: ROC Curves on the True Positive Rates

This section underscores the importance of feature engineering and highlights areas where additional parameters such as "SPACE AND NOTHING" could further improve prediction fidelity.

9.4.4 Sample True vs Predicted

To further evaluate model performance beyond aggregate accuracy scores, sample predictions were analyzed by comparing the true labels against the model's predicted labels and associated probabilities. This analysis provided insight not only into whether the predictions were correct but also into the confidence levels assigned by the model.

Below we will see how the most confused class were behaving with some sample prediction:

Table 2: Sample Image to show the V vs W Conflict

Sample_Image	Prediction for V	Prediction for W	Other prediction
V1115.jpg	99.96%	0.01%	0.03%
V186.jpg	96.74%	3.00%	0.26%
V2922.jpg	99.81%	0.00%	0.19%
V1406.jpg	100%	-	-
V238.jpg	92.10%	6.85%	1.05%
V2237.jpg	98.39%	0.06%	1.55%
V1552.jpg	100%	-	-
V1711.jpg	100%	-	-
V2617.jpg	26.19%	4.86%	68.95%
V2191.jpg	39.88%	59.63%	0.49%

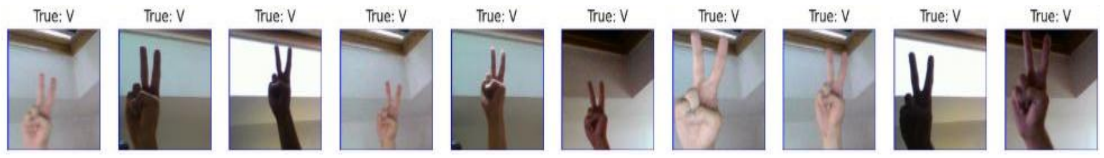


Figure 7: Sample Predictions for V vs W conflict

Overall, the sample true vs predicted analysis demonstrated that the model was not only capable of making highly confident predictions for most gestures but also revealed systematic areas of uncertainty. These insights guided both future data collection (to better cover ambiguous classes) and system design decisions (to handle low-confidence cases gracefully in deployment).

9.5 Final Model Performance

The model development process was promote for model transparency and several features- and performance-based visualizations were generated that can be interpreted for their uses:

9.5.1 Accuracy, Precision, Recall, F1-Score and Confusion Matrix

The Relation to ensure balanced performance across all price types:

a) Accuracy: Measured the overall correctness of predictions across all classes.

b) Precision: Ensured that predicted prices were actually present in the corresponding results.

c) Recall: Checked that all actual price types were successfully captured by the model.

d) F1 Score: Balanced precision and recall to avoid over-fitting to dominant classes.

Table 3: Classification Report of the Trained Model

Character	Precision	Recall	F1-score	Support
A	0.95	1.00	0.97	360
B	1.00	0.98	0.99	360
C	0.99	1.00	1.00	360
D	1.00	0.99	1.00	360
E	0.95	0.98	0.97	360
F	1.00	1.00	1.00	360
G	1.00	0.99	1.00	360
H	0.98	0.99	0.98	360
I	0.99	0.96	0.97	360
J	1.00	0.92	0.96	360
K	1.00	0.94	0.97	360
L	1.00	0.99	1.00	360
M	0.96	1.00	0.98	360
N	0.97	1.00	0.98	360
O	1.00	0.98	0.99	360
P	0.97	1.00	0.98	360

Q	1.00	0.97	0.99	360
R	1.00	0.94	0.97	360
S	0.99	0.91	0.95	360
T	0.96	0.99	0.98	360
U	0.98	0.99	0.98	360
V	0.99	0.85	0.92	360
W	0.85	0.99	0.92	360
X	0.99	0.97	0.98	360
Y	0.90	1.00	0.95	360
Z	1.00	0.98	0.99	360
del	1.00	0.99	1.00	360
Nothing	0.94	1.00	0.97	360
space	1.00	0.99	0.99	360

After running the Data-set we got Macro and Weight avg to be similar and, Model Accuracy was found out to be: 97.58%.
These results indicate a highly reliable model

Table 4: Accuracy and Average of the Classification

Value	Precision	Recall	F1-score	Support
Accuracy	-	-	0.98	10440
Macro avg	0.98	0.98	0.98	10440
Weighted avg	0.98	0.98	0.98	10440

The final ASL gesture recognition model not only achieves state-of-the-art accuracy on the dataset but also generalizes well to unseen data, making it highly suitable for integration into real-time applications.

System Architecture and Implementation

10.1 System Architecture

The architecture of the American Sign Language System is designed to provide seamless integration between the trained deep learning model (MobileNetV2) custom fine-tuning into an interactive Streamlit-based user interface, and auxiliary modules that enhance accessibility and interactivity. The System follows a designed with a modular and scalable architecture, where machine learning inference, user interface rendering, history management, and educational modules are developed the system separates data Preprocessing, model inference, Visualization logic and Rendering into independent layers.

The overall workflow consists of two main phases:

a) **Model Training & Evaluation:** Conducted in Jupyter Notebook using Tensorflow/Keras.

b) **Interactive UI Deployment & Visualization:** Implemented via a Streamlit app for real-time usage.

10.1.1 Model Training Pipeline

The training pipeline is responsible for developing the core gesture recognition model. It is structured into several well-defined stages to ensure reproducibility, robustness, and deployment readiness:

10.1.1 a) Dataset Loading

- The data-set consists of hand gesture images categorized into 29 classes, corresponding to the 26 letters of the ASL alphabet plus space, delete, and nothing.
- Images are organized into directory structures where each sub-folder corresponds to a label, enabling efficient use of ImageDataGenerator.
- Both training and validation sets are balanced to prevent class bias, and a separate test set is reserved for final evaluation.

10.1.1 b) Preprocessing

- Resizing: All input images are resized to 160×160 pixels to match the expected input dimensions of MobileNetV2 while maintaining a trade-off between resolution and computational efficiency.
- Normalization: Pixel values are scaled to the range [-1, 1] to accelerate convergence and stabilize training.
- Data Augmentation: To improve generalization and robustness to hand position/orientation variations.

10.1.1 c) Model Design

- A transfer learning approach is adopted using MobileNetV2 as the base model, chosen for its efficiency and accuracy on mobile/edge devices.
- The base model is initialized with ImageNet weights, excluding the final classification layer.

10.1.1 d) Training Strategy

A two-phase training schedule is followed:

- Feature Extraction Phase: The base (MobileNetV2) layers are frozen, and only the top classifier layers are trained to quickly adapt to ASL gestures.
- Fine-tuning Phase: Selective layers of the backbone are unfrozen, and the model is trained end-to-end with a low learning rate to refine feature representations without catastrophic forgetting.

10.1.1 e) Evaluation

The trained model is assessed using multiple performance metrics:

- Accuracy on validation and test datasets.
- Classification Report including precision, recall, and F1-score for each gesture class.
- Confusion Matrix to identify commonly misclassified gestures, particularly between visually similar signs.

10.1.1 f) Persistence

- The final trained model is exported in HDF5 (.h5) format for Tf/Keras deployment.
- Associated preprocessing artifacts (e.g., normalization scaler, label encoder) are saved as .pkl files to ensure consistency during inference.
- Versioning and model metadata (training date, dataset split, hyperparameters) are logged for reproducibility and potential retraining.

10.1.2 Application Architecture

The second phase delivers an interactive user interface built with Streamlit, enabling both technical and non-technical users to utilize the prediction system effectively.

The application is logically separated into components:

a) main.py: Main application file

b) utils/about.py: To cover the Home Section Content

c) utils/history.py: To compare results and all previous Records

d) utils/live_camera.py: We can predict the Gesture with live web cam

e) utils/sample_gestures.py: Containing a worksheet and the logic of quiz

f) utils/upload_prediction.py: Generating the Result based on user upload.

g) utils/word_maker.py: To make a word using live camera.

These are the files used in Frontend and more Backend files to support and connect the logic.

10.1.3 UI Logic And Flow

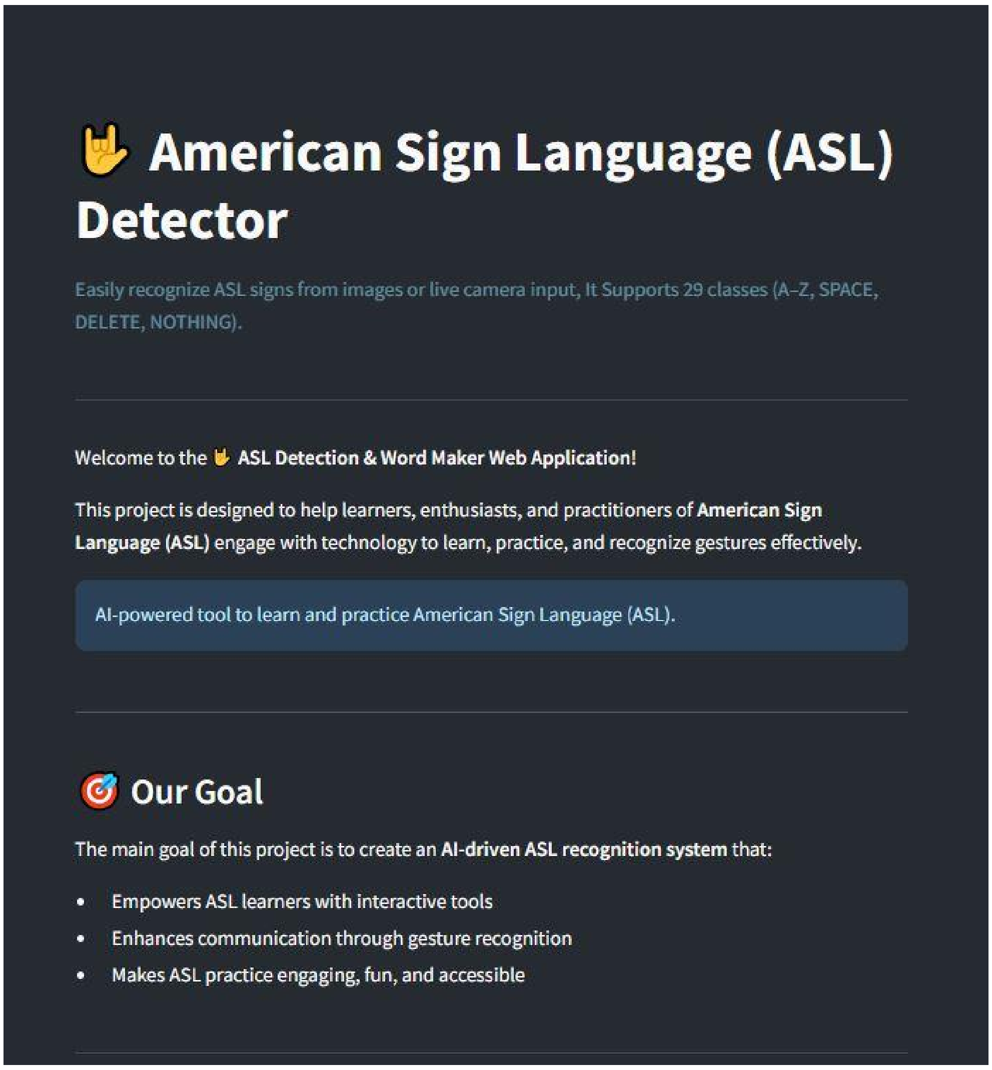
The user experience of the Streamlit GUI is designed for intuitive navigation:

a) On launch, users can greeted with the home page which contains all the information that they need to understand how it works and how to navigate the page.

b) On the Sidebar, User can find different options to navigate through different features, along with the tips for each page.

c) Upload Prediction, The User can upload an image and the Model will provide them with the Top 5 Predictions.
d) Live Detection: The Model can detect and can predict live prediction of the Gesture the user makes every second.
e) Word Maker: We can select any Number of words and then the user has to input them to make the word.
f) History: It saves all the Interactions of the Predictions made by the user.
g) Sample Gesture: It has Sample Gestures with Descriptions and A game to guess the Sign Symbols

An Example of the Streamlit UI made for better UX of the users who are unfamiliar:



🤟 American Sign Language (ASL) Detector

Easily recognize ASL signs from images or live camera input, It Supports 29 classes (A–Z, SPACE, DELETE, NOTHING).

Welcome to the 🤟 **ASL Detection & Word Maker Web Application!**

This project is designed to help learners, enthusiasts, and practitioners of **American Sign Language (ASL)** engage with technology to learn, practice, and recognize gestures effectively.

AI-powered tool to learn and practice American Sign Language (ASL).

🎯 Our Goal

The main goal of this project is to create an **AI-driven ASL recognition system** that:

- Empowers ASL learners with interactive tools
- Enhances communication through gesture recognition
- Makes ASL practice engaging, fun, and accessible

Figure 8: User Interface of the Application

10.2 Application Working and Processing

The user interface is built using Streamlit, a Python-based framework ideal for building data-driven apps.

10.2.1 How it Works

The application in functionality of the primary goal is organized into three primary tabs:

a) Upload Prediction: For parsing a uploaded image by the user to get the Expected Output for the Character.

b) Live Detection: For the user who wants to Capture or to Detect the Prediction in Real time.

c) Word Maker: If the user wants to make a word with the letters detected and captured by the model it is used to make words and possibly a meaning of that word.

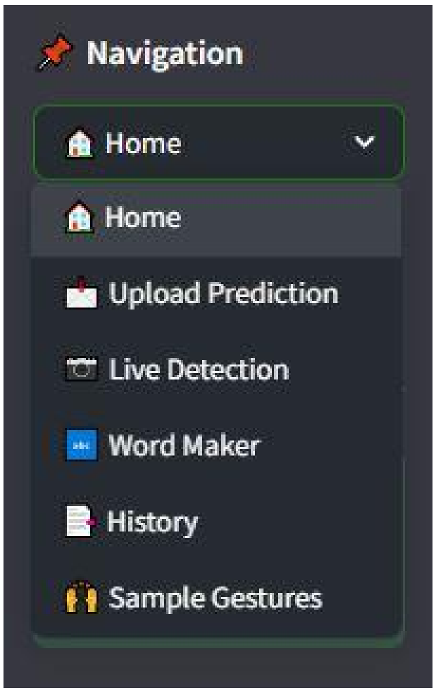
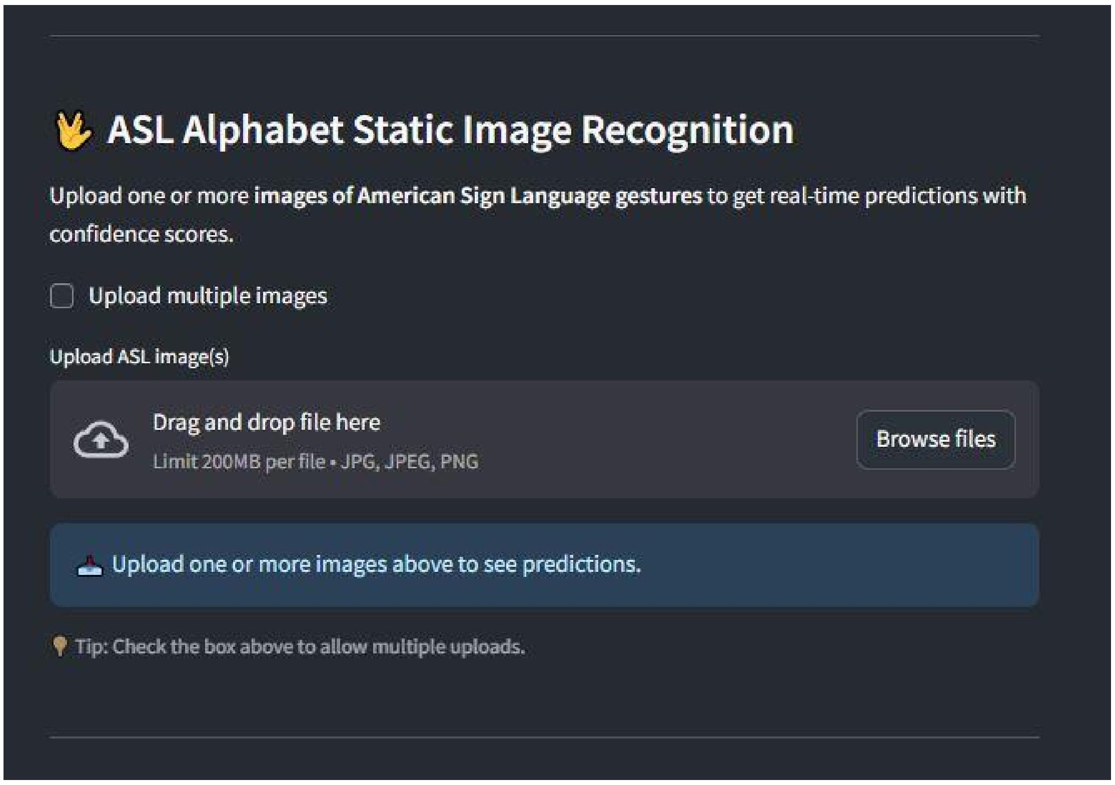


Figure 9: Navigation Sidebar for Different Features

10.2.2 Upload Prediction

The Upload Prediction module enables users to interact with the trained gesture recognition model by uploading gesture images for inference. This feature is designed for flexibility, accuracy, and usability, supporting both single and batch predictions.

a) Upload File: Users can upload one or more gesture images in .jpg or .jpeg through a file uploader interface, A checkbox option is provided to toggle between single-upload and multi-file upload modes.



🖖 ASL Alphabet Static Image Recognition

Upload one or more **images of American Sign Language gestures** to get real-time predictions with confidence scores.

☐ Upload multiple images

Upload ASL image(s)

Drag and drop file here
Limit 200MB per file • JPG, JPEG, PNG

Browse files

📤 Upload one or more images above to see predictions.

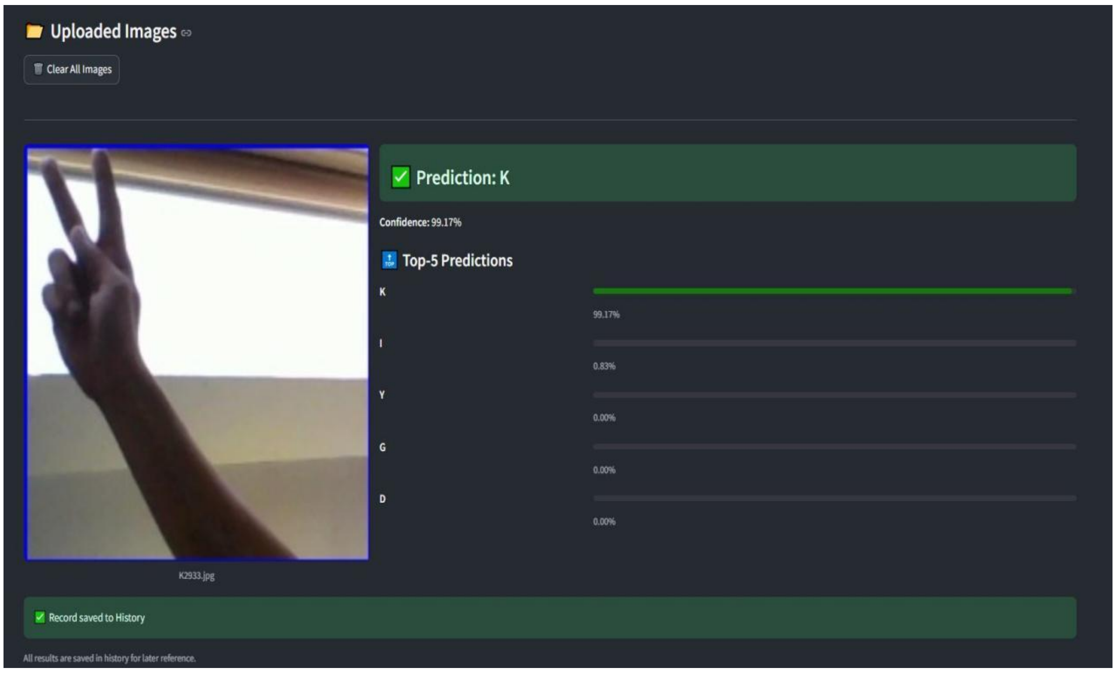
💡 Tip: Check the box above to allow multiple uploads.

Figure 10: Uploading Prediction Interface

b) Preprocessing and Prediction: Upon Uploading, each images undergoes the preprocesing pipeline (resizing to 160x160, normilization to [-1,1]), The Model then generates top 5 predictions of most probable classes and since the model is accurate, most predictionw will be ranked correctly.

c) Batch Uploads and Pagination: For multi-file uploads user can select the option and when the preprocessing is complete the result will be displayed with pagination, allowing to keep track of the Uploads and avoid Overcrowding.

d) Result Display: Every prediction result includes Top-5 Prediction classes sorted by probability along with Confidence Score in Percentage, A visual confirmation of the uploaded image next to the predicted image and output.



Uploaded Images

Clear All Images

Prediction: K

Confidence: 99.17%

Top-5 Predictions

K 99.17%

I 0.83%

Y 0.00%

G 0.00%

D 0.00%

K2933.jpg

Record saved to History

All results are saved in history for later reference.

Figure 11: Prediction of an uploaded Image

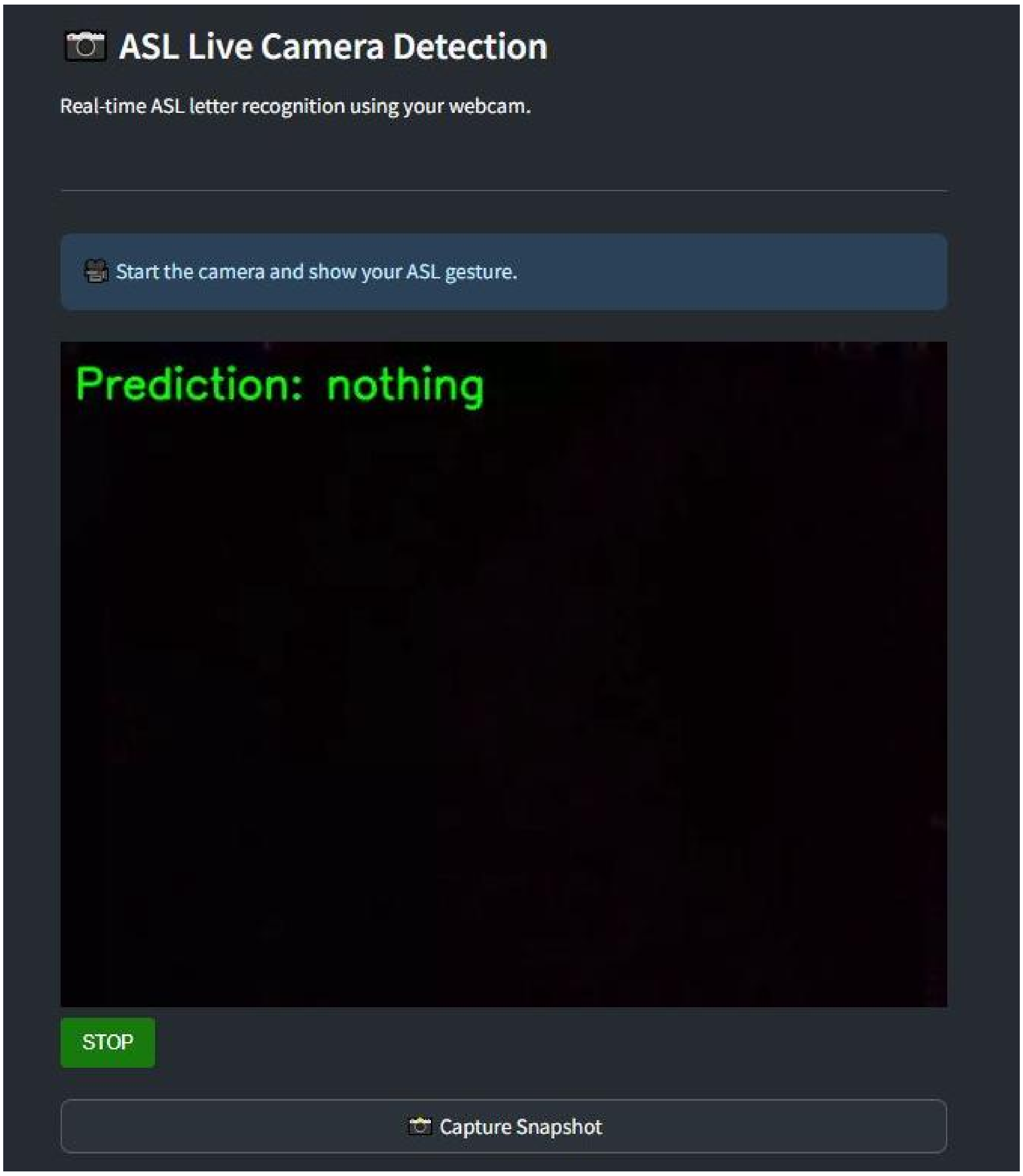
10.2.3 Live Prediction

The Live Prediction module enables real-time gesture recognition through a camera interface, powered by Streamlit WebRTC. This feature allows users to interact with the system more naturally by performing gestures directly in front of their camera rather than uploading static images.

a) Camera Integration: Users can select a camera device and then by clicking Start, the system activates the camera and begins streaming in real time, then the user can position their hand gestures.

b) Real-Time Prediction: Each frame is passed through the model, so that the predictions on the top-left side updates continuously.

c) Snapshot Capture: Clicking the Snapshot button will click the current frame, and the Model will so more details such as top 5 predictions along with the captured frame that was clicked.


ASL Live Camera Detection

Real-time ASL letter recognition using your webcam.

Start the camera and show your ASL gesture.

Prediction: nothing

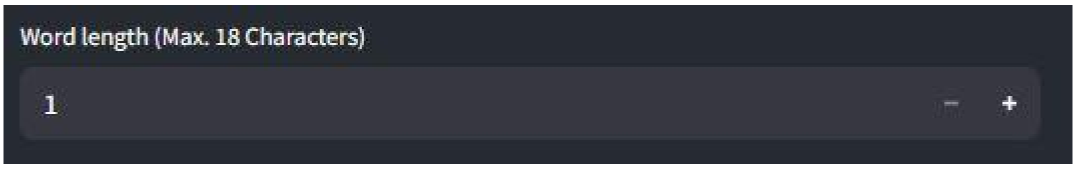
Figure 12: Live Prediction with Frame Upgradation

d) Saving and Retake Options: After capturing, the system saves the prediction with a confirmation and then the user can use the Retake Button to take the next picture.

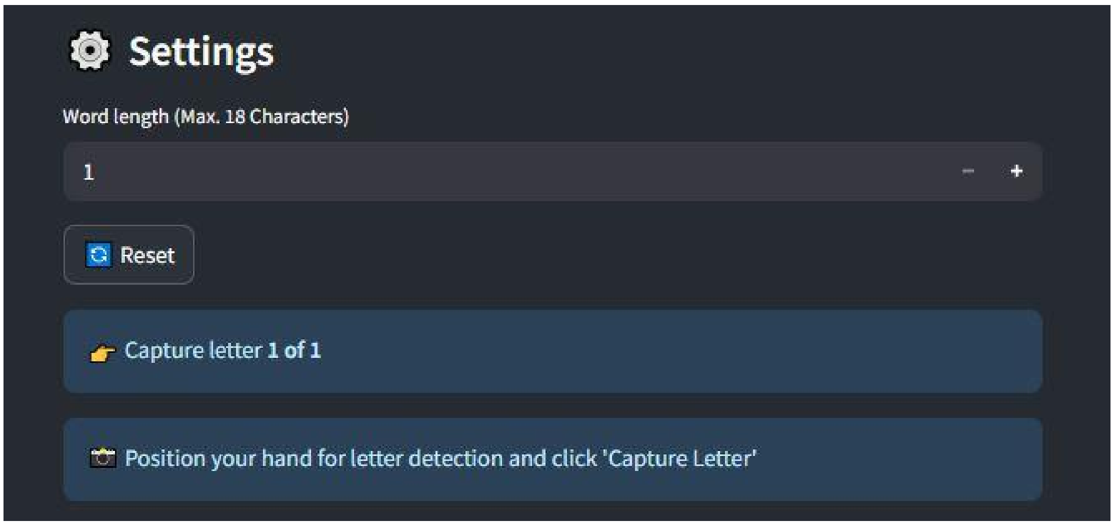
10.2.4 Word Maker

The Word Maker module extends gesture recognition beyond single letters by allowing users to sequentially capture multiple gestures and combine them into complete words. This feature leverages live camera input for interactive word-building and integrates with external dictionary resources for meaning lookup.

a) Word Length Selection: Users begin by selecting the desired word length, The Parameter ensures overflowing and easiler to work with the user.


Word length (Max. 18 Characters)
1

b) Sequential Capture: Once the word length is set, the user activates the camera feed, the system allows user to capture a series of image(s), After each capture the image displayed on interface along with Top Prediction results.


Settings
Word length (Max. 18 Characters)
1

Capture letter 1 of 1
Position your hand for letter detection and click 'Capture Letter'

Figure 13: Selection of Length and the Description

c) Word Formation: When required number of images has been captured, the letters combined in words that will be displayed prominently, enabling the user to validate the recognition outcome.

d) Dictionary Integration: The completed word is then checked with a valid meaning through a Dictionary API, If the word is found it will display the meaning below, Else the system classifies it as a gibbreish word.

10.3 User Interface Design and Features

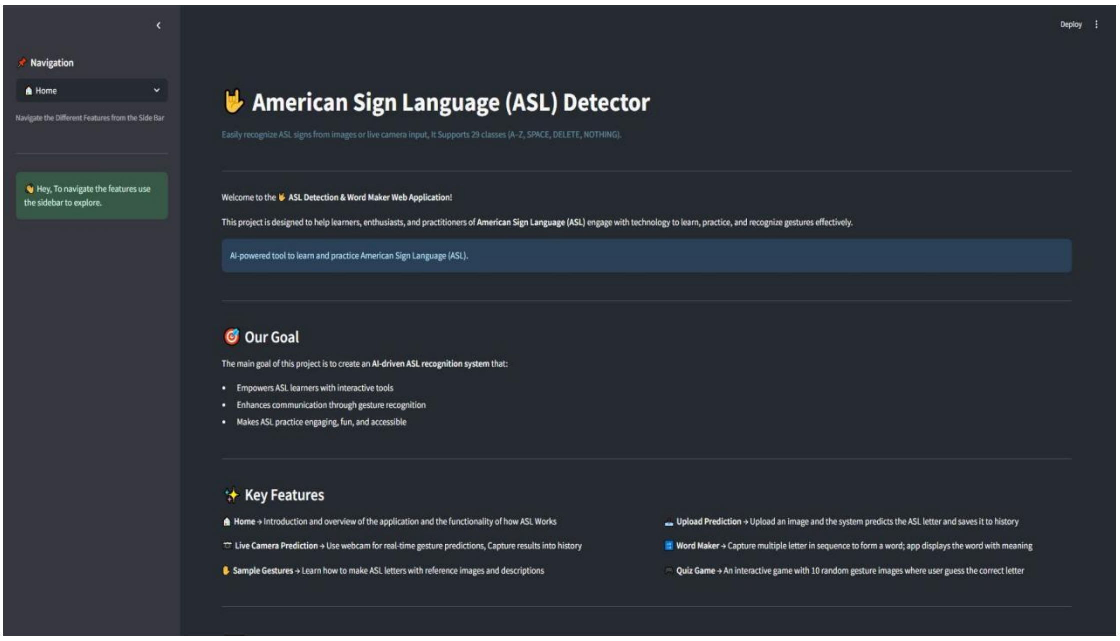
The Streamlit Application provides multiple tabs, accessible via a navigation sidebar.

10.3.1 Overview

The Home Page serves as the entry point to the ASL Detection & Word Maker web application. It provides users with an overview of the project, its goals, and its key functionalities, while also offering intuitive navigation to all major features.

The home page begins with a warm welcome message that introduces the ASL Detection & Word Maker application. A concise project description explains the purpose: enabling learners, enthusiasts, and practitioners of American Sign Language (ASL) to engage with AI-powered gesture recognition tools for education and communication. A brief highlight emphasizes that the system is an AI-driven tool for practicing and recognizing ASL gestures interactively.

The sidebar provides persistent navigation links to all modules, making it easy for users to switch between features. Below the navigation, a tips section dynamically updates either every 60 seconds or upon page refresh, offering quick hints and guidance to enhance the user experience. A footer tip (st.info) reinforces the availability of key features such as Live Detection.



Navigate the Different Features from the Side Bar

👋 Hey, To navigate the features use the sidebar to explore.

🤟 American Sign Language (ASL) Detector

Easily recognize ASL signs from images or live camera input, It Supports 29 classes (A-Z, SPACE, DELETE, NOTHING).

Welcome to the 🤟 **ASL Detection & Word Maker Web Application!**

This project is designed to help learners, enthusiasts, and practitioners of **American Sign Language (ASL)** engage with technology to learn, practice, and recognize gestures effectively.

AI-powered tool to learn and practice American Sign Language (ASL).

🎯 Our Goal

The main goal of this project is to create an **AI-driven ASL recognition system** that:

- Empowers ASL learners with interactive tools
- Enhances communication through gesture recognition
- Makes ASL practice engaging, fun, and accessible

✨ Key Features

🏠 **Home** → Introduction and overview of the application and the functionality of how ASL Works

📷 **Live Camera Prediction** → Use webcam for real-time gesture predictions, Capture results into history

✋ **Sample Gestures** → Learn how to make ASL letters with reference images and descriptions

📤 **Upload Prediction** → Upload an image and the system predicts the ASL letter and saves it to history

🔠 **Word Maker** → Capture multiple letter in sequence to form a word; app displays the word with meaning

🎮 **Quiz Game** → An interactive game with 10 random gesture images where user guess the correct letter

Figure 14: Overview of the Interface

10.3.2 Gestures Reference

The Gesture Reference module functions as a learning aid, providing users with a complete visual guide to the ASL alphabet set. It allows learners to understand and practice hand gestures in a structured and accessible way, reinforcing the predictions made by the recognition system.

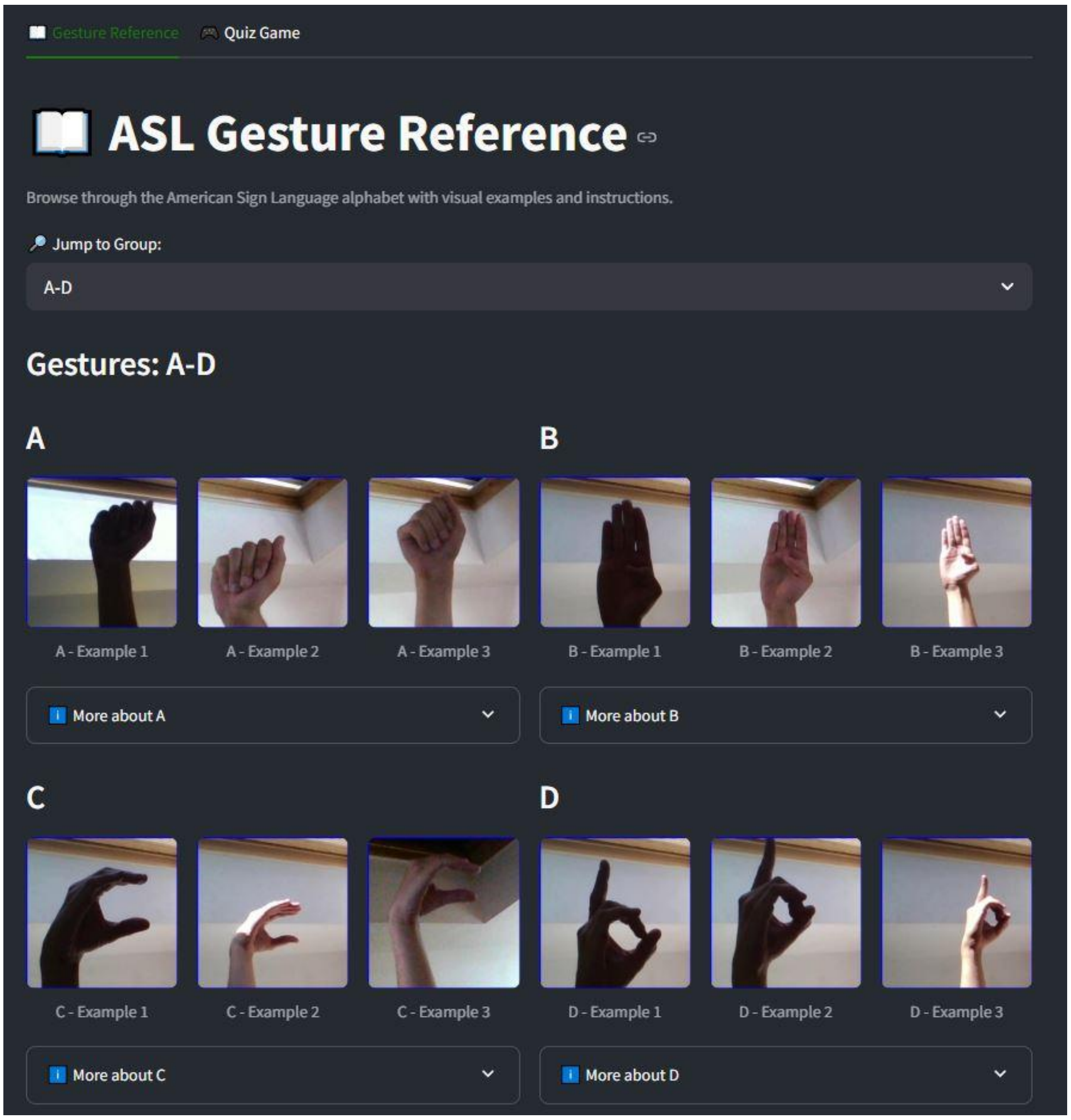


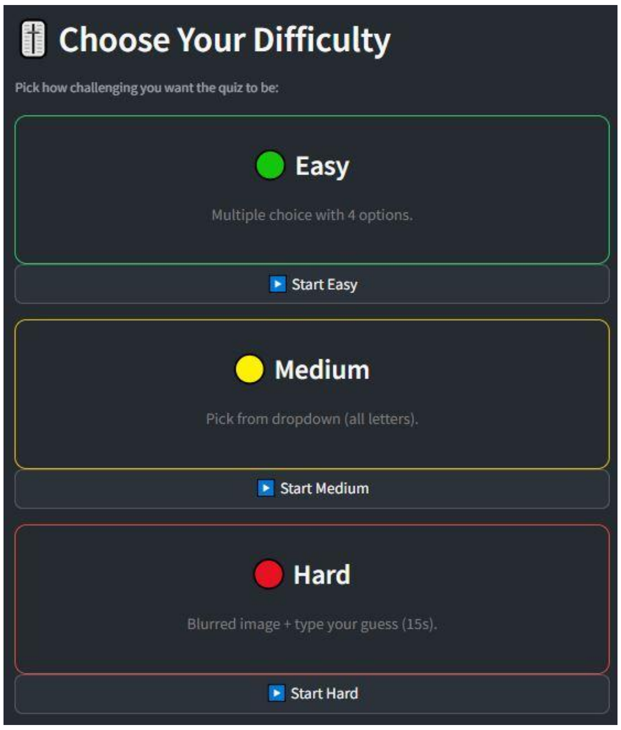
Figure 15: Sample Gestures for Reference

10.3.3 Gestures Learning Quiz

The Gesture Quiz module provides an interactive self-assessment tool for users to test their knowledge of ASL gestures. It combines random gesture image prompts with different difficulty levels, score tracking, and detailed feedback to create an engaging learning experience.

a) Quiz Setup: The game consists of 10 rounds, each selects a randomly selected image, User must identify the correct symbol from the options, A difficulty selector is displayed at the start, offering three challenging levels:

i. Easy - Multiple-choice with 4 options
ii. Medium - Full dropdown of all options
iii. Hard - Blurred image, Reveal image for only 15 seconds and to type the answer


Choose Your Difficulty

Pick how challenging you want the quiz to be:

🟢 Easy

Multiple choice with 4 options.

▶ Start Easy

🟡 Medium

Pick from dropdown (all letters).

▶ Start Medium

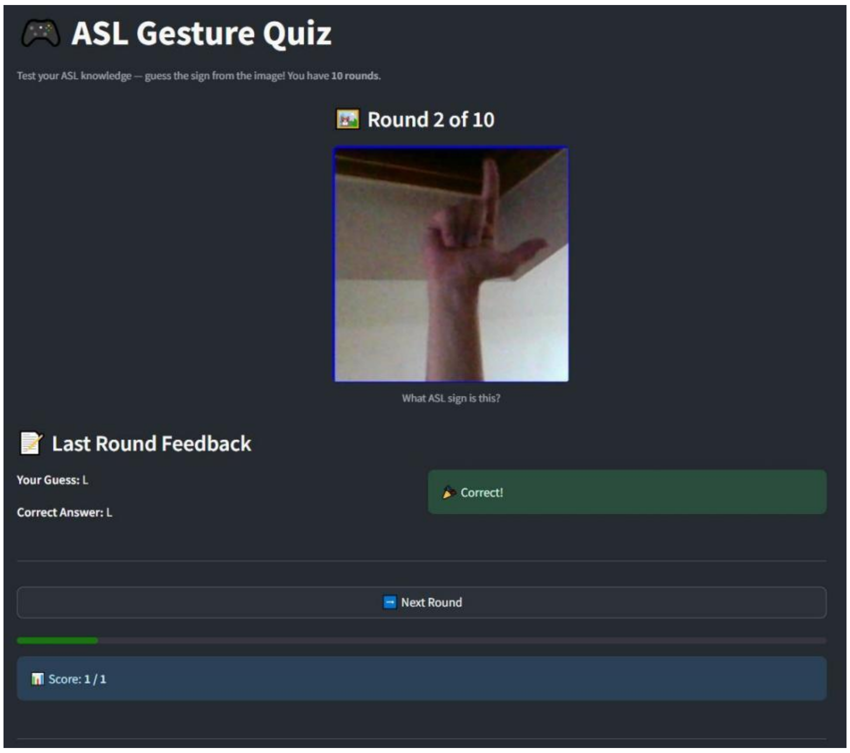
🔴 Hard

Blurred image + type your guess (15s).

▶ Start Hard

Figure 16: Difficulty Selector

b) Quiz Flow: At the Start of each round, ASL gesture image is displayed, Clicking Submit locks the answers, Correct Increases the score and Incorrect does not do anything, The user receives the feedback after submitting.


ASL Gesture Quiz

Test your ASL knowledge — guess the sign from the image! You have **10 rounds**.

Round 2 of 10

What ASL sign is this?

Last Round Feedback

Your Guess: L

Correct Answer: L

Correct!

Score: **1 / 1**

Figure 17: Preview of Gesture Quiz

c) Score Tracking: A progress bar shows at the bottom during the round and updates after each input from the user, The feedback keeps the experience interactive.

d) End-of-Quiz Summary: When it ends, the final scorecard will display the final score, along with the Gesture Image from each round, User Guess and Correct Answers indicting each if they got it right or wrong.

e) Exporting Result: PDF report of the quiz is generated after the Summary is displayed, It includes Final score, Pre-round Details. The user can also press the reply button to play again.

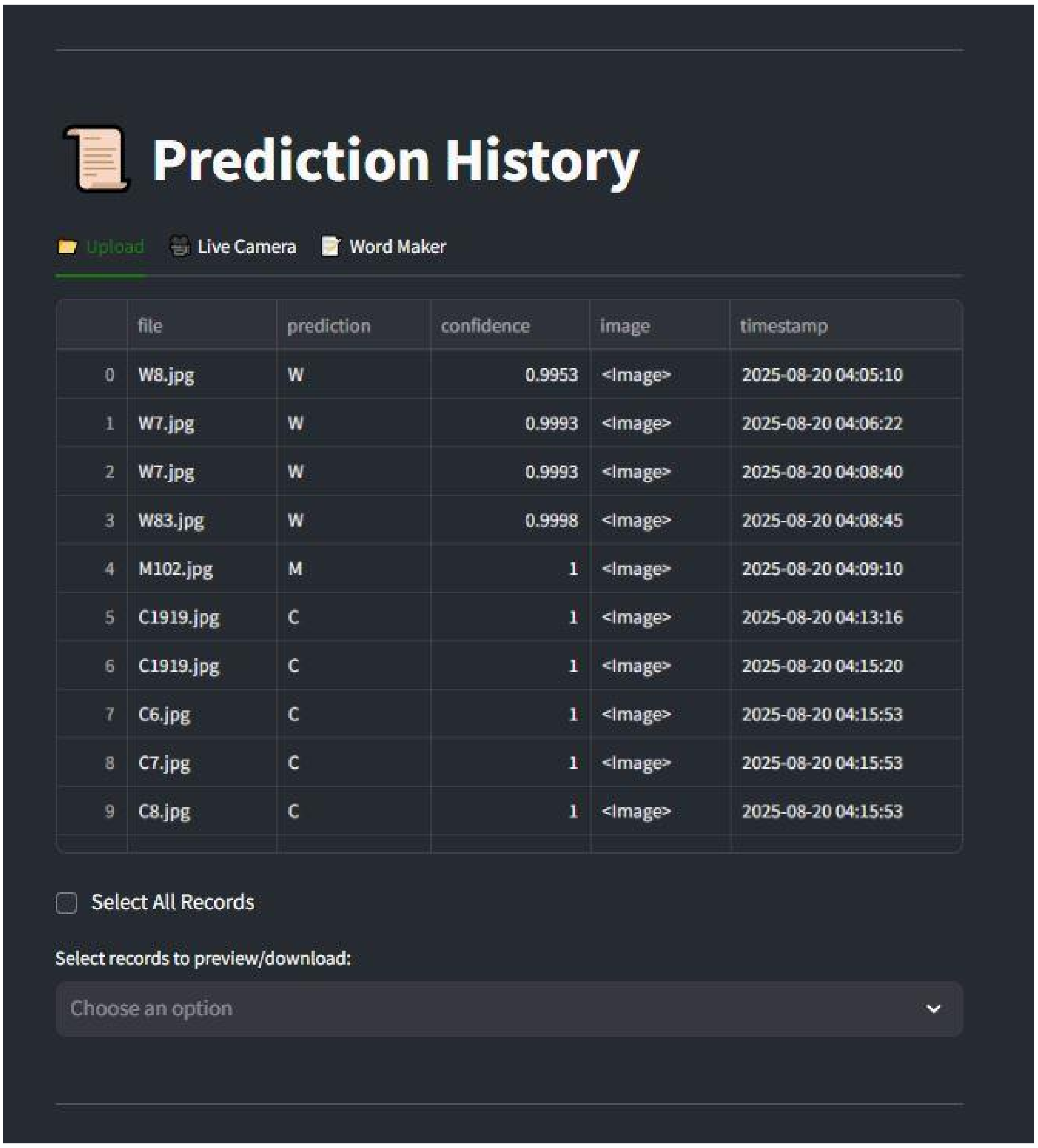
10.3.4 History Management

The system maintains a comprehensive prediction history that covers all modules — Uploads, Live Camera Captures, Word Maker, and Quiz Game. This history allows users to review past sessions, preview captured data, and download them in different formats for record-keeping.

a) Centralized Logging: Every prediction is automatically stored in the history.json file, Associated with images saving on dedicated folders.

b) Tabbed History View: The history interface is organized into 3 main tab:

i. Upload: Shows all uploaded Image Prediction
ii. Live Camera: Displays captures taken during live prediction
iii. Word Maker: Stores sequential letter captures and generated words.



Prediction History

	file	prediction	confidence	image	timestamp
0	W8.jpg	W	0.9953	<Image>	2025-08-20 04:05:10
1	W7.jpg	W	0.9993	<Image>	2025-08-20 04:06:22
2	W7.jpg	W	0.9993	<Image>	2025-08-20 04:08:40
3	W83.jpg	W	0.9998	<Image>	2025-08-20 04:08:45
4	M102.jpg	M	1	<Image>	2025-08-20 04:09:10
5	C1919.jpg	C	1	<Image>	2025-08-20 04:13:16
6	C1919.jpg	C	1	<Image>	2025-08-20 04:15:20
7	C6.jpg	C	1	<Image>	2025-08-20 04:15:53
8	C7.jpg	C	1	<Image>	2025-08-20 04:15:53
9	C8.jpg	C	1	<Image>	2025-08-20 04:15:53

☐ Select All Records

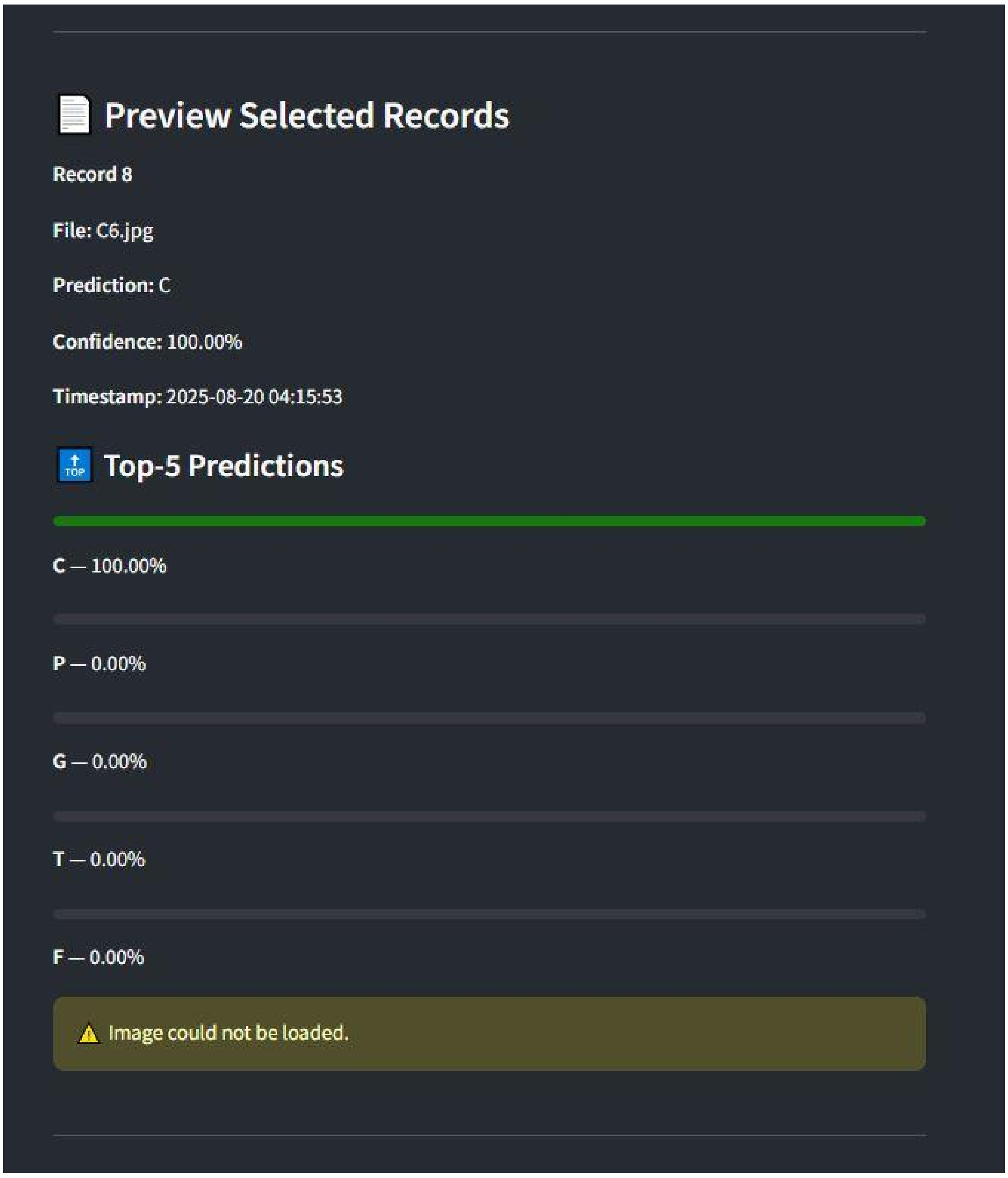
Select records to preview/download:

Choose an option

Figure 18: Selection of Data History

Each tab presents the records in a data table view, making it easy to scan through past sessions.

c) Detailed Record Preview: Users can select one, mulitple or all records to Preview, The Details Displays File Name, Top 5 Prediction, Confidence, Timestamp, and Image if available. For Word Maker it also includes final word and its meaning along with each captured image.


Preview Selected Records

Record 8

File: C6.jpg

Prediction: C

Confidence: 100.00%

Timestamp: 2025-08-20 04:15:53

Top-5 Predictions

C — 100.00%

P — 0.00%

G — 0.00%

T — 0.00%

F — 0.00%

⚠ Image could not be loaded.

Figure 19: Preview of Saved Prediction

d) Export and Download: The User can download the selected and Previewed data in various formats such as CSV, JSON and PDF.

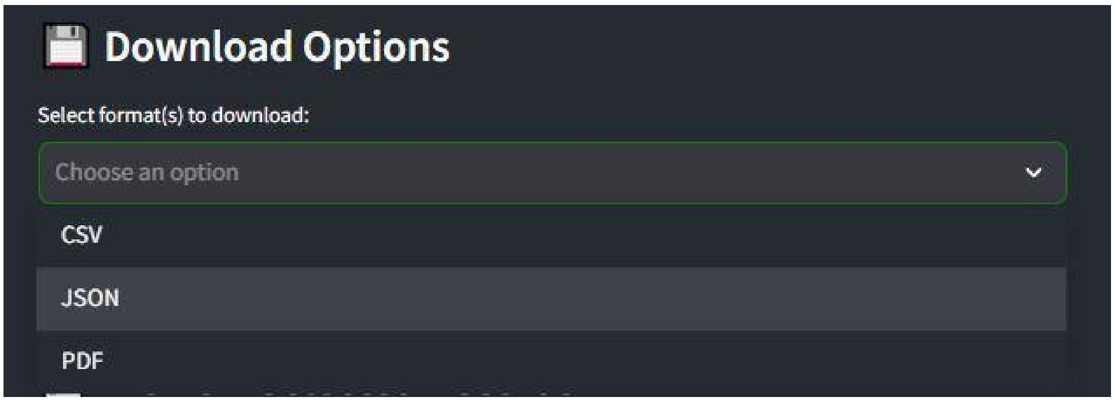


Figure 20: Options for Downloading Data

10.3.5 User Interface Design

The application's interface is built with clarity, accessibility, and usability as core priorities, ensuring both beginners and advanced users can navigate and interact with the system effectively, Some of the key Features are:

a) Navigation Sidebar

b) Responsive Layouts

c) Visual Feedback

d) Accessibility

e) Consistency Across Modules

10.4 Access the Program

To Run the Program it can be done using a single line bash command:

```
>> streamlit run main.py
```

or

Open the file:

`Run.bat` in the root folder

Alternatively, it supports deployment on Streamlit Cloud, Heroku, or Docker. Model loading is optimized with @st.cache_resource to avoid repeated reloads, making inference efficient.

Project Concept and Conclusion

11.1 Project Concept

The American Sign Language (ASL) Recognition System was conceptualized with the primary objective of bridging the communication gap between hearing-impaired individuals and the broader community. Traditional communication methods often depend on human interpreters, written transcripts, or manual teaching aids, which, while effective, have limitations in terms of scalability, accessibility, and real-time interaction.

This project introduces a novel solution by leveraging deep learning and computer vision to recognize ASL gestures in real time and convert them into meaningful textual outputs. The system not only performs recognition but also provides a user-friendly application interface, making it suitable for both assistive communication and educational purposes.

Unlike standalone classifiers, this system integrates multiple modules into a cohesive platform:

- Upload-Based Recognition – Users can upload one or multiple images for prediction.
- Live Camera Detection – Real-time gesture recognition powered by WebRTC.
- Word Maker – Sequential gesture capturing for meaningful word formation.
- Gesture Reference Library – Visual dictionary of ASL symbols with descriptions.
- Interactive Quiz Game – Self-assessment tool to reinforce learning.
- History Management – All predictions are logged, previewed, and exportable in multiple formats.

By combining these modules, the system functions as a dual-purpose platform, An assistive tool for communication and learning platform for ASL beginners and educators.

11.2 Visualization and Performance

Interpretability and transparency were central to the design. The application provides clear visual aids alongside predictions:

- Confidence Bar Chart – Displays probability distribution across all 29 classes.
- Top-5 Predictions Panel – Lists alternative gestures with descending confidence levels.
- Session History Table – Tracks user inputs, outputs, and timestamps.
- Quiz Analytics – Summarizes user performance by difficulty level.
- Gesture Reference Grid – Provides multiple labeled examples per gesture for learning.
- Word Maker Output – Sequentially predicted letters combined into words with dictionary meaning lookup.

The system maintained real-time responsiveness, with prediction latency averaging <1 second per frame during live camera detection.

11.3 Benefits and Limitations

The American Sign Language Prediction System brings several impactful advantages:

a) **User-Friendly Interface**: Simple yet powerful design allowing intuitive inputs like sliders, drop-down and toggle switches.

b) **Multi-Modal Inputs**: Accepts static images as wella s real-time video streaming.

c) **Accessibility**: Web-based deployment enables usage without requiring complex installations.

d) **Educational Values**: Built-in reference library for the user to check anytime.

e) **History Tracking**: Session data exportable as PDF, CSV and JSON.

f) **Extensibility:** Sample Modular architecture enables future extensions and more.

These features made the system highly suitable for the end user.

However, the system also has some limitations:

a) **Dataset Constraints:** The system is trained on the Unified Provided Dataset, which may reduce real-world adaptability.

b) **Environmental Sensitivity:** Accuracy may degrade under poor lighting or cluttered background.

c) **Security Constraints:** Due to the Security concerns making this work on a local Network won't work as the Camera permission needs dedicated private and trusted servers.

These limitations provide direction for future improvements and scalability considerations. Nevertheless, it serves as a strong proof-of-concept and fulfil the objective for an intelligent Prediction systems.

11.4 Achievements and Conclusion

11.4.1 Achievements

This project demonstrates a complete deep learning lifecycle implementation, from data ingestion and model training to web deployment and end-user interactivity. Notable accomplishments include:

a) Successfully trained and fine tuned a high-accuracy MobileNetV2 Model.

b) Developed a St interface with functional tabs covering: Live Predictions and Upload Predictions.

c) Incorporated and implemented real-time gesture recognition with WebRTC integration.

d) Extended beyond recognition with Word Maker and Quiz Game modules.

e) A Modular, Maintainable and Extendable Architecture.

f) Enabled multi-format session export for usability and record-keeping.

11.4.2 Use Case

The system has potential applications in several domains:

a) **Assisitive Technology:** Enabling communication for hearing-impaired individuals.

b) **Data Science Project:** Acts as a template for regression-based predictive UI app..

c) **Retailers:** Could be a prototype for the Mobile price analytics tools and inspiration.

d) **Educational Demo:** Ideal project to demonstrate a beginner in the feature engineering, model deployment, machine learning training phases and user-centered designing.

11.4.3 Conclusion

The ASL Recognition System is a successful integration of deep learning, computer vision, and human-centered design. By offering real-time recognition, structured learning resources, and historical tracking, it proves both the technical feasibility and social value of gesture recognition technology.

Future improvements such as dynamic gesture recognition, speech synthesis, and larger, more diverse datasets will allow the system to evolve into a comprehensive sign language interpreter, advancing inclusivity and accessibility in communication technologies.

Reference

[1] Simonyan, K., & Zisserman, A. (2015). "Very Deep Convolutional Networks for Large-Scale Image Recognition." International Conference on Learning Representations (ICLR).

[2] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). "MobileNetV2: Inverted Residuals and Linear Bottlenecks." IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).

[3] American Sign Language (ASL) Dataset – Kaggle. Available at: https://www.kaggle.com/datasets/grassknoted/asl-alphabet.

[4] Scikit-learn Documentation,
https://scikit-learn.org/stable/

[5] Streamlit Documentation,
https://docs.streamlit.io/

[6] Pandas Library Documentation,
https://pandas.pydata.org/docs/

[7] Matplotlib and Seaborn – Data Visualization Tools,
https://matplotlib.org/
https://seaborn.pydata.org/

[8] Chollet, F. et al. (2015). Keras: Deep Learning for Python. Available at: https://keras.io/

[9] Python Official Documentation,
https://numpy.org/doc/

[10] Numpy Documation,
https://elevenlabs.io/

[11] Zhang, Z., Yin, Z., Chen, J., & Niu, Z. (2016). "American Sign Language Recognition Based on Deep Convolutional Neural Networks." IEEE International Conference on Software Engineering and Service Science (ICSESS).

[12] Rastgoo, R., Kiani, K., & Escalera, S. (2020). "Sign Language Recognition: A Deep Survey." Expert Systems with Applications, 164, 113794.

[13] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … & Duchesnay, É. (2011). "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research, 12, 2825–2830.

[14] TensorFlow https://www.tensorflow.org/about