

# PROJECT DEVELOPMENT PHASE

DIVYANSH JAIN

MUDIT AGRAWAL

KANISHK PRASAD

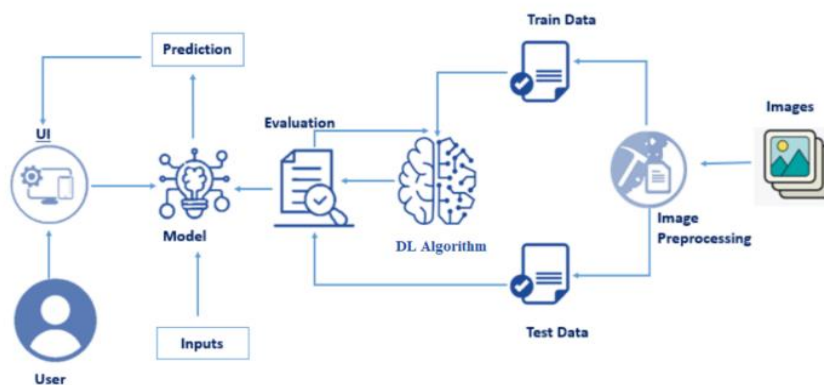
EBI JOHN SINJIN

## ASL (American Sign Language) - Alphabet Image recognition

### Introduction:

American Sign Language (ASL) serves as the primary language for deaf individuals in North America, relying on visual communication through hand gestures, facial expressions, and body movements. Recent years have witnessed a growing interest in creating technologies to bridge the communication divide between the deaf and hearing communities. One such innovation is ASL Alphabet Image Recognition, a project involving the training of a machine learning model for the classification of images depicting hand signs representing the 26 letters of the English alphabet, along with three extra categories for the signs "space," "delete," and "nothing." This trained model has the potential to be applied in real-time video streams to recognize the ASL alphabet, facilitating improved communication between the deaf and hearing communities.

### Technical Architecture:



## **Prerequisites:**

For the successful completion of our ASL image recognition project, you'll need the following software, concepts, and packages:

- Python and its data science and machine learning libraries, which we'll primarily manage using Anaconda Navigator, a free and open-source distribution.
- Conda, an open-source package management system.
- Deep Learning Concepts: Focusing on Convolutional Neural Networks (CNN), a specialized neural network class widely used for visual image analysis.
- Flask: A popular Python web framework that will aid in building our web application.
- Tools like Google Colab and Visual Studio Code will be employed for this project.

## **Project Objectives:**

By the end of this project, you will achieve the following:

- Acquire a strong grasp of Convolutional Neural Network principles and techniques.
- Develop a comprehensive understanding of image data.
- Master data preprocessing techniques, including resizing, normalization, and data splitting.
- Gain proficiency in creating a web application using the Flask framework.

## **Project Workflow:**

Our project will unfold as follows:

1. User Interaction: The user interacts with the UI to select an image.
2. Image Analysis: The chosen image will be processed by the CNN model integrated into the Flask application.
3. Prediction Display: The CNN model will analyze the image and showcase its prediction on the Flask UI.

To accomplish this, we will carry out the following tasks:

- Data Collection: Gather or download the dataset necessary for training the CNN model, specifically focused on ASL image recognition.
- Data Preprocessing: Prepare the dataset by resizing, normalizing, and splitting it into training and testing sets.
- Model Building:
  - a. Import the required Python libraries for constructing the CNN model.
  - b. Define the input shape of the image data for ASL signs.
  - c. Add model layers:
    - i. Convolutional Layers: Employ filters to extract distinctive features from the input ASL images.
    - ii. Pooling Layers: Reduce the spatial dimensions of the extracted features.
    - iii. Fully Connected Layers: Flatten the output from convolutional layers and employ fully connected layers to classify ASL signs.
  - d. Compile the model by specifying the optimizer, loss function, and metrics for training.

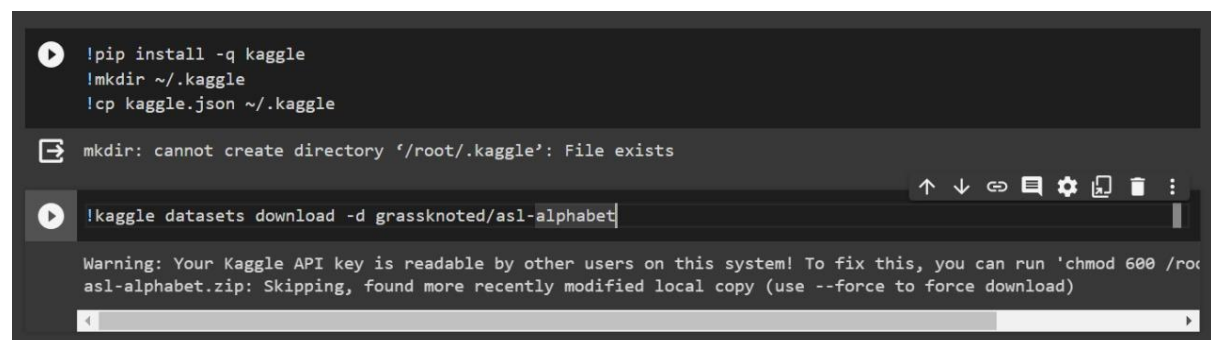
- Model Training: Train the model using the training dataset, utilizing the ImageDataGenerator class to augment ASL images and monitoring accuracy on the validation set to prevent overfitting.
- Model Evaluation: Assess the trained model's performance on the testing dataset, calculating accuracy and other relevant metrics.
- Model Deployment: Save the model for future use and prepare it for real-world applications, facilitating ASL sign recognition.

## Project Structure:

### MILE STONE 1: DATA COLLECTION

All necessary links had been uploaded into GITHUP repo collaboration.

### Setting up the Kaggle API



```
!pip install -q kaggle
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle

mkdir: cannot create directory '/root/.kaggle': File exists

!kaggle datasets download -d grassknotted/asl-alphabet

Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'.
asl-alphabet.zip: Skipping, found more recently modified local copy (use --force to force download)
```

## Download and unzip data

```
!kaggle datasets download -d grassknotted/asl-alphabet

Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'. Skipping, found more recently modified local copy (use --force to force download)

!unzip /content/asl-alphabet.zip

Streaming output truncated to the last 5000 lines.
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing19.jpg
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing190.jpg
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1900.jpg
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1901.jpg
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1902.jpg
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1903.jpg
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1904.jpg
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1905.jpg
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1906.jpg
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1907.jpg
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1908.jpg
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing1909.jpg
inflating: asl_alphabet_train/asl_alphabet_train/nothing/nothing191.jpg
```

## MILESTONE 2: DATA PREPARATION

### Installing necessary packages

```
# import data processing and visualisation libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# import image processing libraries
import cv2
import skimage
from skimage.transform import resize

# import tensorflow and keras
import tensorflow as tf
from tensorflow import keras
import os

print("Packages imported...")

Packages imported... Installing necessary packages
```

## Label Configuration

```
[ ] batch_size = 64
    imageSize = 64
    target_dims = (imageSize, imageSize, 3)
    num_classes = 29

    train_len = 87000
    train_dir = '/content/asl_alphabet_train/asl_alphabet_train'

    def get_data(folder):
        X = np.empty((train_len, imageSize, imageSize, 3), dtype=np.float32)
        y = np.empty((train_len,), dtype=np.int)
        cnt = 0
        for folderName in os.listdir(folder):
            if not folderName.startswith('.'):
                if folderName in ['A']:
                    label = 0
                elif folderName in ['B']:
                    label = 1
                elif folderName in ['C']:
                    label = 2
                elif folderName in ['D']:
                    label = 3
                elif folderName in ['E']:
                    label = 4
                elif folderName in ['F']:
                    label = 5
```

```
                elif folderName in ['F']:
                    label = 5
                elif folderName in ['G']:
                    label = 6
                elif folderName in ['H']:
                    label = 7
                elif folderName in ['I']:
                    label = 8
                elif folderName in ['J']:
                    label = 9
                elif folderName in ['K']:
                    label = 10
                elif folderName in ['L']:
                    label = 11
                elif folderName in ['M']:
                    label = 12
                elif folderName in ['N']:
                    label = 13
                elif folderName in ['O']:
                    label = 14
                elif folderName in ['P']:
                    label = 15
                elif folderName in ['Q']:
                    label = 16
                elif folderName in ['R']:
                    label = 17
                elif folderName in ['S']:
                    label = 18
                elif folderName in ['T']:
                    label = 19
                elif folderName in ['U']:
```

```

        label = 19
    elif folderName in ['U']:
        label = 20
    elif folderName in ['V']:
        label = 21
    elif folderName in ['W']:
        label = 22
    elif folderName in ['X']:
        label = 23
    elif folderName in ['Y']:
        label = 24
    elif folderName in ['Z']:
        label = 25
    elif folderName in ['del']:
        label = 26
    elif folderName in ['nothing']:
        label = 27
    elif folderName in ['space']:
        label = 28
    else:
        label = 29
    for image_filename in os.listdir(folder + '/' + folderName):
        img_file = cv2.imread(folder + '/' + folderName + '/' + image_filename)
        if img_file is not None:
            img_file = skimage.transform.resize(img_file, (imageSize, imageSize, 3))
            img_arr = np.asarray(img_file).reshape((-1, imageSize, imageSize, 3))

            X[cnt] = img_arr
            y[cnt] = label
            cnt += 1

    return X,y
X_train, y_train = get_data(train_dir)
print("Images successfully imported...")

```

## MILESTONE 3: DATA PREPROCESSING

This code creates metadata for the ASL alphabet dataset by getting all the image files for each label,

creating a list of image paths, and a corresponding list of labels. It then creates a Pandas dataframe

using these two lists with columns "image\_path" and "label".

```

[ ] print("The shape of X_train is : ", X_train.shape)
    print("The shape of y_train is : ", y_train.shape)

The shape of X_train is : (87000, 64, 64, 3)
The shape of y_train is : (87000,)

[ ] print("The shape of one image is : ", X_train[0].shape)

The shape of one image is : (64, 64, 3)

```

```
[ ] X_data = X_train
    y_data = y_train
    print("Copies made...")

Copies made...

[ ] from sklearn.model_selection import train_test_split

    X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.3, random_state=42, stratify=y_data)

from tensorflow.keras.utils import to_categorical
y_cat_train = to_categorical(y_train, 29)
y_cat_test = to_categorical(y_test, 29)

[ ] # Checking the dimensions of all the variables
    print(X_train.shape)
    print(y_train.shape)
    print(X_test.shape)
    print(y_test.shape)
    print(y_cat_train.shape)
    print(y_cat_test.shape)

(60900, 64, 64, 3)
(60900,)
(26100, 64, 64, 3)
(26100,)
(60900, 29)
(26100, 29)
```

## Data Augmentation:

This code defines a function named `data_augmentation()` which generates image data generators using the `ImageDataGenerator` class from the Keras library. The function defines three generators for training, validation, and testing datasets respectively using the `flow_from_dataframe()` method of the `ImageDataGenerator` class. The generators take the image path and label information from dataframes `data_train`, `data_val`, and `data_test` and apply the same data augmentation technique of rescaling the image pixel values to a range of 0 to 1. The batch size, image size, and number of classes are also defined using the CFG class attributes.



```

] # Data Augmentation (Just Rescale)
def data_augmentation():
    datagen = ImageDataGenerator(rescale=1/255.,)
    # Training Dataset
    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 Dataset
    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),
    )

seed_everything(2023)
train_generator, validation_generator, test_generator = data_augmentation()

```

```

Found 58103 validated image filenames belonging to 29 classes.
Found 15847 validated image filenames belonging to 29 classes.
Found 13050 validated image filenames belonging to 29 classes.

```

## MILESTONE 4: MODEL BUILDING

### Packages for Model Building

```

[ ] from keras.models import Sequential
    from keras.layers import Conv2D, MaxPooling2D, Activation, Dense, Flatten
    print("Packages imported...")

```

```

Packages imported...

```

```

▶ model = Sequential()

model.add(Conv2D(32, (5, 5), input_shape=(64, 64, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(29, activation='softmax'))

model.summary()

```

➡ Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 60, 60, 32)	2432
activation (Activation)	(None, 60, 60, 32)	0
max_pooling2d (MaxPooling2D)	(None, 30, 30, 32)	0

```

[ ] from tensorflow.keras.callbacks import EarlyStopping
    early_stop = EarlyStopping(monitor='val_loss',patience=2)

```

```

[ ] model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

```

## Fitting the data in model

```
model.fit(X_train, y_cat_train,
          epochs=10,
          batch_size=64,
          verbose=2,
          validation_data=(X_test, y_cat_test),
          callbacks=[early_stop])
```

Epoch 1/10  
952/952 - loss: 1.0934 - accuracy: 0.6723 - val\_loss: 0.2868 - val\_accuracy: 0.9092 - 453s/epoch - 476ms/step  
Epoch 2/10  
952/952 - loss: 0.1583 - accuracy: 0.9484 - val\_loss: 0.1066 - val\_accuracy: 0.9659 - 481s/epoch - 505ms/step  
Epoch 3/10  
952/952 - loss: 0.0783 - accuracy: 0.9752 - val\_loss: 0.1939 - val\_accuracy: 0.9477 - 481s/epoch - 505ms/step  
Epoch 4/10  
952/952 - loss: 0.0494 - accuracy: 0.9846 - val\_loss: 0.0536 - val\_accuracy: 0.9830 - 475s/epoch - 499ms/step  
Epoch 5/10  
952/952 - loss: 0.0445 - accuracy: 0.9857 - val\_loss: 0.0317 - val\_accuracy: 0.9884 - 470s/epoch - 494ms/step  
Epoch 6/10  
952/952 - loss: 0.0304 - accuracy: 0.9908 - val\_loss: 0.0407 - val\_accuracy: 0.9874 - 472s/epoch - 496ms/step  
Epoch 7/10  
952/952 - loss: 0.0374 - accuracy: 0.9888 - val\_loss: 0.0476 - val\_accuracy: 0.9837 - 478s/epoch - 502ms/step  
<keras.src.callbacks.History at 0x7bdf9271a710>

## MILESTONE 5: MODEL EVALUATION

```
[ ] metrics = pd.DataFrame(model.history.history)
print("The model metrics are")
metrics
```

The model metrics are

	loss	accuracy	val_loss	val_accuracy
0	1.093370	0.672299	0.286788	0.909157
1	0.158349	0.948424	0.106573	0.965900
2	0.078258	0.975172	0.193878	0.947663
3	0.049385	0.984581	0.053560	0.982988
4	0.044546	0.985665	0.031719	0.988391
5	0.030379	0.990788	0.040670	0.987433
6	0.037448	0.988752	0.047627	0.983716

```
[ ] from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test, predictions.argmax(axis=1)))
```

	precision	recall	f1-score	support
0	1.00	0.97	0.99	900
1	0.96	0.99	0.98	900
2	0.99	1.00	1.00	900
3	0.98	1.00	0.99	900
4	1.00	0.95	0.97	900
5	0.99	1.00	1.00	900
6	0.99	1.00	0.99	900
7	1.00	1.00	1.00	900
8	0.98	1.00	0.99	900
9	1.00	0.99	1.00	900
10	1.00	0.91	0.95	900
11	0.99	1.00	1.00	900
12	0.96	1.00	0.98	900
13	0.99	0.99	0.99	900
14	0.99	0.99	0.99	900
15	0.98	1.00	0.99	900
16	1.00	0.97	0.99	900
17	0.92	1.00	0.96	900
18	1.00	0.96	0.98	900
19	1.00	0.97	0.99	900
20	1.00	0.89	0.94	900
21	0.89	0.99	0.93	900
22	1.00	0.99	1.00	900
23	0.98	0.99	0.98	900
24	0.98	0.99	0.99	900
25	1.00	1.00	1.00	900
26	1.00	1.00	1.00	900
27	1.00	1.00	1.00	900
28	1.00	0.99	1.00	900
accuracy			0.98	26100

## MILESTONE 6: LOAD AND TEST THE MODEL

Predicting on the model

```
[ ] model.evaluate(X_test,y_cat_test,verbose=0)
```

```
[0.04762706905603409, 0.9837164878845215]
```

```
[ ]
```

```
predictions = model.predict(X_test)
print("Predictions done...")
```

```
816/816 [=====] - 51s 63ms/step
Predictions done...
```

```
▶ print(predictions)
```

```
→ [[2.47299443e-15 9.00875187e-37 4.88014410e-37 ... 2.97706522e-31
    0.00000000e+00 0.00000000e+00]
 [2.82475605e-11 2.41463750e-13 3.19870946e-06 ... 9.99815226e-01
 2.08218977e-14 8.53608029e-09]
 [6.00669155e-05 6.97072480e-08 3.64633598e-22 ... 6.28011196e-18
 6.59798321e-28 2.74841270e-14]
 ...
 [5.36439586e-07 2.56411670e-09 5.22070722e-14 ... 9.13543963e-10
 1.94733818e-17 2.77476317e-08]
 [3.21985674e-08 2.98178787e-10 9.76235449e-17 ... 6.43449053e-14
 1.85970132e-14 7.11963485e-16]
 [9.65894476e-10 2.72212919e-10 3.27400052e-11 ... 1.01599392e-06
 9.99998450e-01 1.11380265e-13]]
```

```

model = tf.keras.models.load_model('/content/asl_vgg16_best_weights.h5')
# Load the test image
image_path = '/content/asl-alphabet/asl_alphabet_test/asl_alphabet_test/H_test.jpg'
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (64, 64))

# Preprocess the image
img = tf.keras.applications.mobilenet_v2.preprocess_input(img)

# Make predictions on the image
predictions = model.predict(np.array([img]))

# Get the predicted class label
predicted_class = labels[np.argmax(predictions)]

print(f"The predicted class is {predicted_class}")

```

```

1/1 [=====] - 0s 246ms/step
The predicted class is H

```

```

# Load the test image
image_path = '/content/asl-alphabet/asl_alphabet_test/asl_alphabet_test/nothing_test.jpg'
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img, (64, 64))

# Preprocess the image
img = tf.keras.applications.mobilenet_v2.preprocess_input(img)

# Make predictions on the image
predictions = model.predict(np.array([img]))

# Get the predicted class label
predicted_class = labels[np.argmax(predictions)]
|
print(f"The predicted class is {predicted_class}")

```

```

WARNING:tensorflow:5 out of the last 2904 calls to <function Model.make_predict_function.<locals>.predic
1/1 [=====] - 0s 414ms/step
The predicted class is nothing

```

## Milestone 7: Application Building

Now that our model is trained, we'll proceed to develop a Flask application with a user interface that runs in our local browser. In this Flask application, we'll gather input parameters from an HTML page. These parameters will be passed to the model for predicting the type of garbage, and the results will be displayed on the HTML page to inform the user. When the user interacts with the UI and selects the



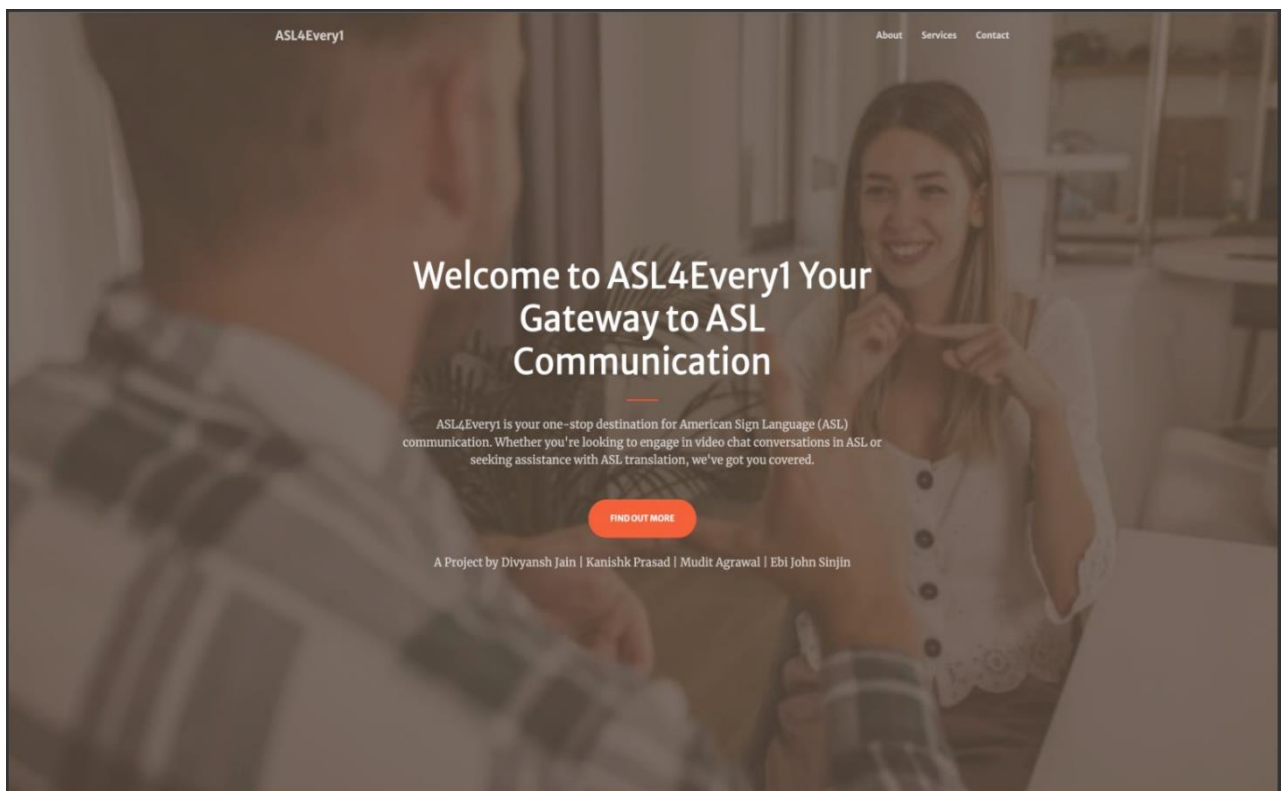
"Image" button, they will be directed to the next page where they can choose an image for prediction.

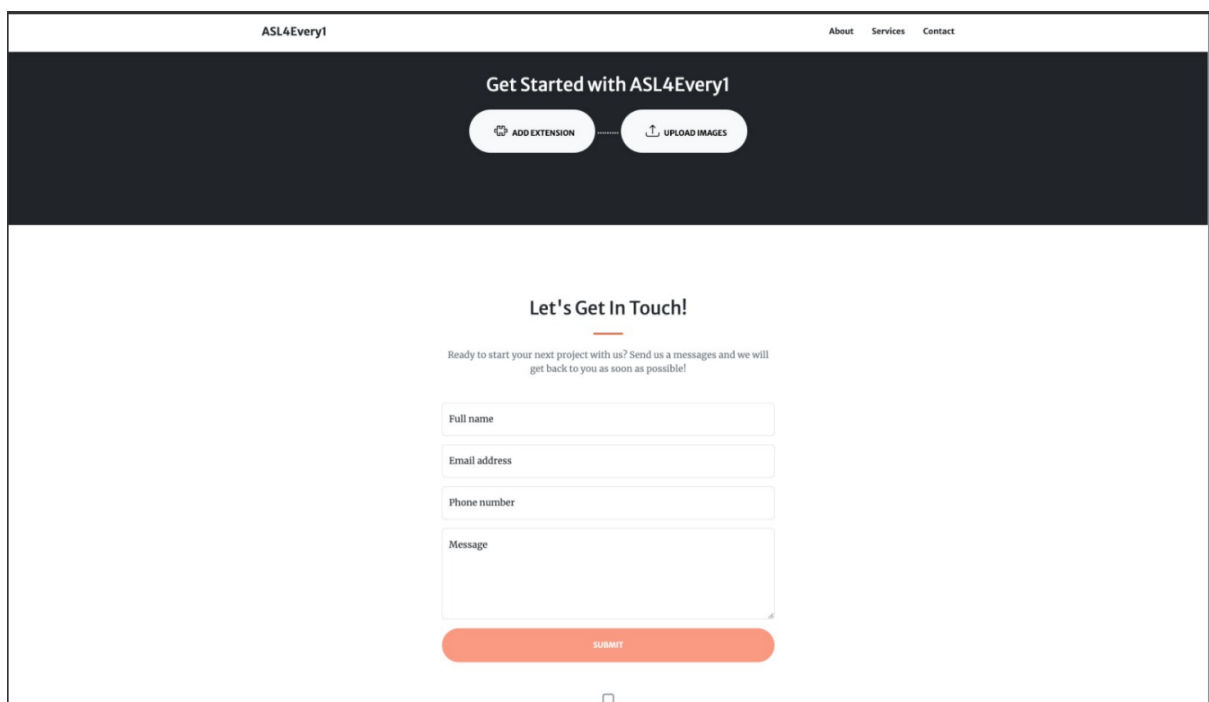
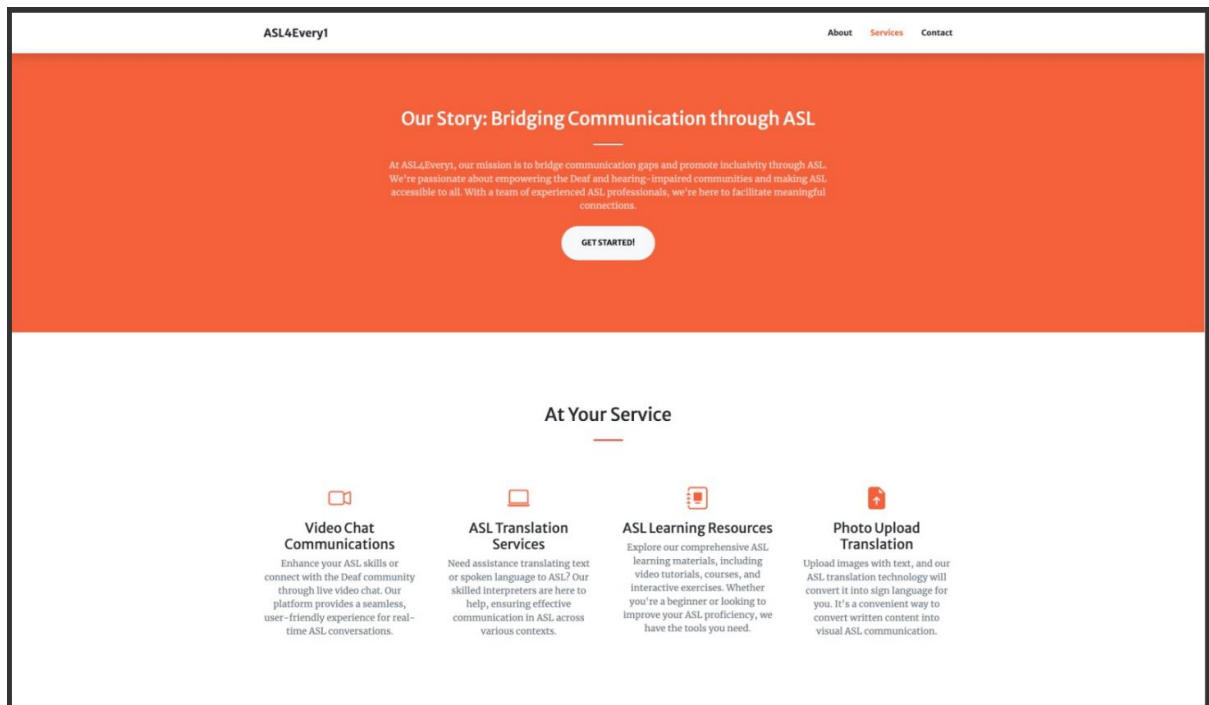
To achieve this, we will undertake the following activities:

### Activity 1: Create HTML Pages

- We will utilize HTML to construct the front-end components of our web application.
- In this context, we will generate three HTML pages: index.html, prediction.html, and logout.html.
- index.html will serve as the homepage.
- prediction.html will be responsible for displaying the prediction page.
- logout.html will provide the result of the prediction.

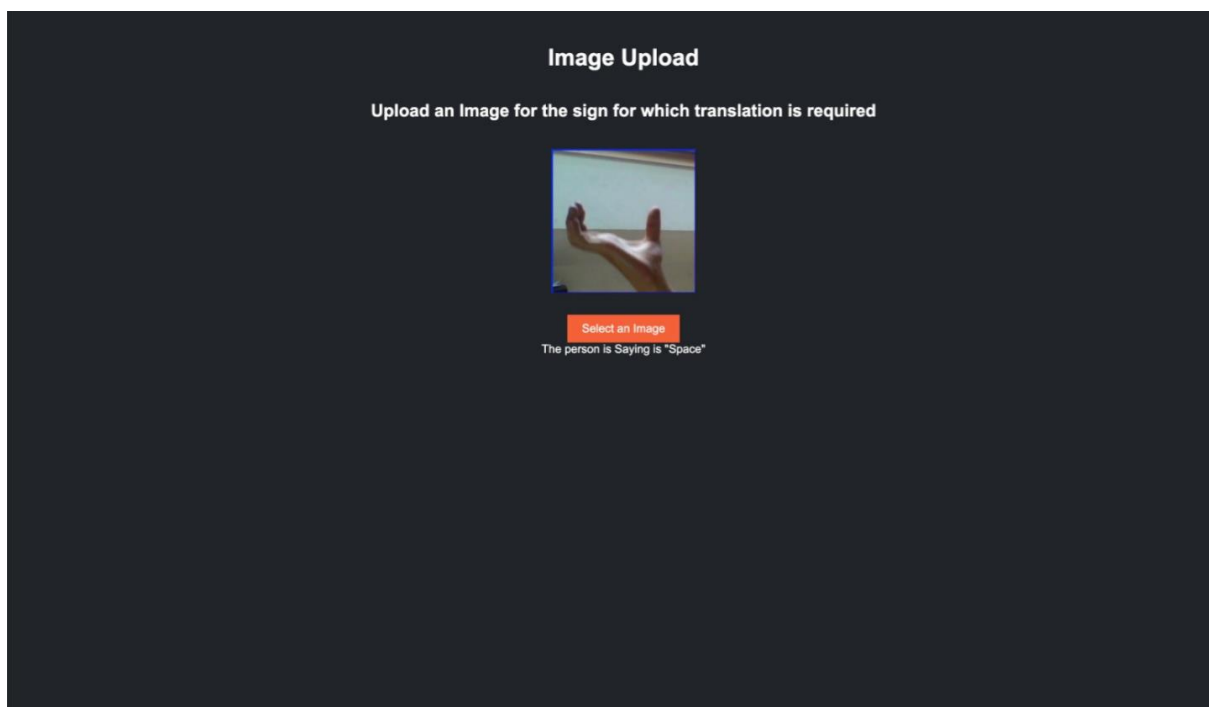
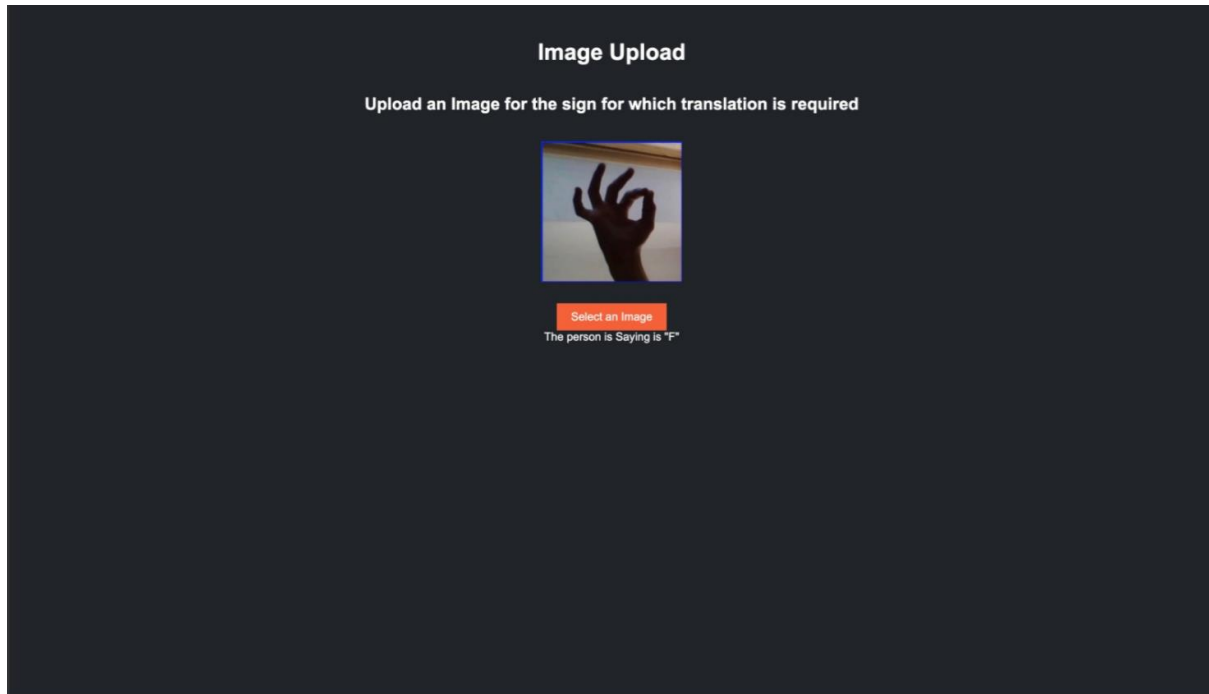
**Below mentioned is our website for ASL recognition made by our team and have all required functionalities**







After doing all the required tasks and now we can see the main working as we upload the images our model recognizes the signs as shown below



## Image Upload

Upload an Image for the sign for which translation is required



Select an Image

The person is Saying is "K"

## Image Upload

Upload an Image for the sign for which translation is required



Select an Image

The person is Saying is "Delete"

This would be the prediction on the model

All rest important codes, flask components are uploaded in GITHUB

```
[ ] model.evaluate(X_test,y_cat_test,verbose=0)

[0.04762706905603409, 0.9837164878845215]

[ ]
predictions = model.predict(X_test)
print("Predictions done...")

816/816 [=====] - 51s 63ms/step
Predictions done...

▶ print(predictions)

➞ [[2.47299443e-15 9.00875187e-37 4.88014410e-37 ... 2.97706522e-31
0.00000000e+00 0.00000000e+00]
[2.82475605e-11 2.41463750e-13 3.19870946e-06 ... 9.99815226e-01
2.08218977e-14 8.53608029e-09]
[6.00669155e-05 6.97072480e-08 3.64633598e-22 ... 6.28011196e-18
6.59798321e-28 2.74841270e-14]
...
[5.36439586e-07 2.56411670e-09 5.22070722e-14 ... 9.13543963e-10
1.94733818e-17 2.77476317e-08]
[3.21985674e-08 2.98178787e-10 9.76235449e-17 ... 6.43449053e-14
1.85970132e-14 7.11963485e-16]
[9.65894476e-10 2.72212919e-10 3.27400052e-11 ... 1.01599392e-06
9.99998450e-01 1.11380265e-13]]
```

To launch the Flask application, simply follow these instructions:

1. Go to the project directory using your terminal or command prompt.
2. Run the command "python app.py" in the terminal to initiate the Flask server.
3. After starting the server, you can access the web application by opening your web browser and navigating to the local host address.

4. Once the application loads, you can upload an image and submit the form.

5. The application will then utilize the trained model to predict the plant species in the image.

6. The prediction outcome will be presented on the web page.

By adhering to these steps, you can interact with the Flask application, upload images, and obtain plant species predictions.

THANK YOU