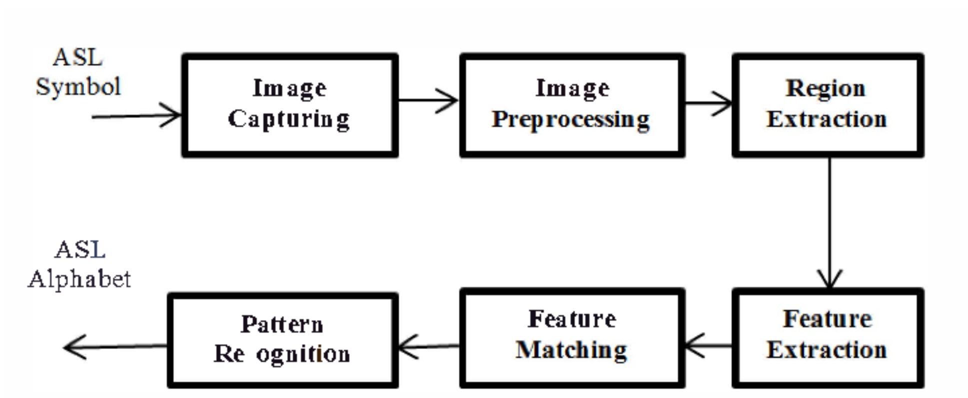
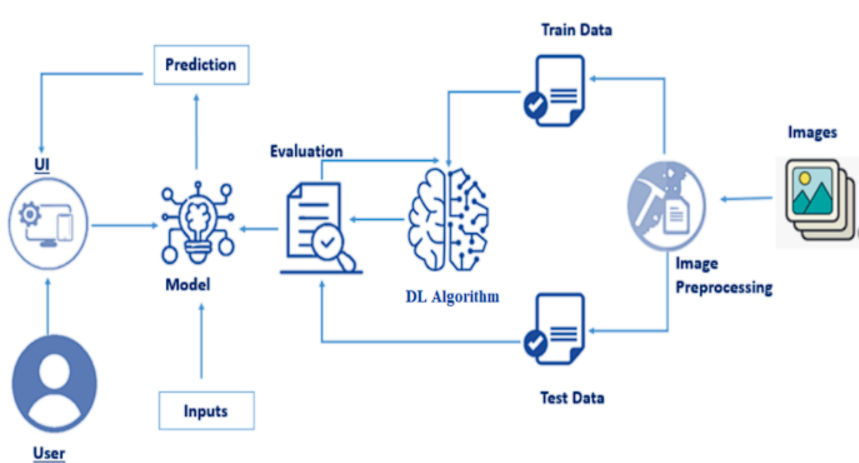
Introduction:
The American Sign Language (ASL) is the primary language used by deaf individuals in North America. It is a visual language that uses a combination of hand gestures, facial expressions, and body
movements to convey meaning. In recent years, there has been an increasing interest in developing
technologies to help bridge the communication gap between the deaf and hearing communities. One such technology is ASL Alphabet Image Recognition, which is an image classification task that
aims to recognize the ASL alphabet from images of hand signs. This project involves training a machine learning model to classify images of hand signs corresponding to the 26 letters of the English
alphabet, as well as three additional classes for the signs for "space", "delete", and "nothing". The trained model can be used to develop applications that can recognize the ASL alphabet from
real-time video streams, which could be used to improve communication between the deaf and hearing communities.

Technical Architecture:



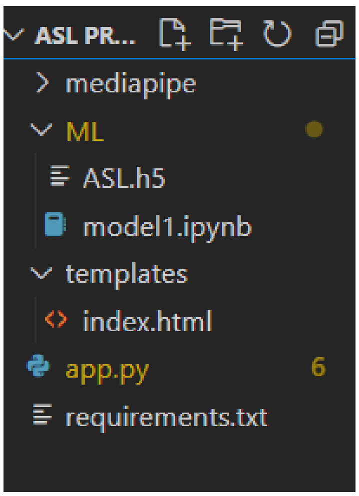
Prerequisites:
To complete this project, you must require the following software's, concepts, and packages
Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, VScode, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Google collab and VS code
● Deep Learning Concepts
o CNN: a convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery.
CNN Basic
o Mediapipe- MediaPipe is an open-source framework for recognizing the hands in a live video feed.
● Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.
● HTML, CSS- web frameworks used for creating web pages.

Project Objectives:
By the end of this project you will:
● Know fundamental concepts and techniques of Convolutional Neural Network.
● Gain a broad understanding of image data.
● Know how to pre-process/clean the data using different data preprocessing techniques.
● know how to build a web application using the Flask framework.

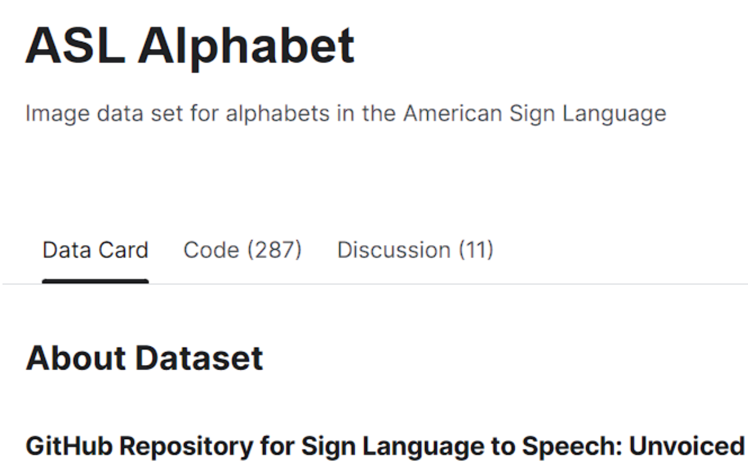
Project Flow:
● The user interacts with the web UI (User Interface) using a live video feed.
● The hands are recognized by using mediapipe and as the capture button is pressed the gesture is captured
● The chosen image analyzed by the model which is integrated with flask application.
● CNN Models analyze the image, then prediction is showcased on the Flask UI.
To accomplish this, we have to complete all the activities and tasks listed below
● Data Collection: Collect or download the dataset that you want to train your CNN on.
● Data Preprocessing: Preprocess the data by resizing, normalizing, and splitting the data into training and testing sets.
● Model Building:
a. Import the necessary libraries for building the CNN model
b. Define the input shape of the image data
c. Add layers to the model:
i. Convolutional Layers: Apply filters to the input image to create feature maps
ii. Pooling Layers: Reduce the spatial dimensions of the feature maps

iii. Fully Connected Layers: Flatten the output of the convolutional layers and apply fully connected layers to classify the images
d. Compile the model by specifying the optimizer, loss function, and metrics to be used during training
● Model Training: Train the model using the training set with the help of the ImageDataGenerator class to augment the images during training. Monitor the accuracy of the model on the validation set to avoid overfitting.
● Model Evaluation: Evaluate the performance of the trained model on the testing set. Calculate the accuracy and other metrics to assess the model's performance.
● Model Deployment: Save the model for future use and deploy it in real-world applications.

Project Structure:
Create a Project folder which contains files as shown below




MILE STONE 1: DATA COLLECTION
The dataset used is from Kaggle. Download the zip file and extract the image dataset and open it in Jupyter notebook or as an ipynb file in VS Code
https://www.kaggle.com/datasets/grassknoted/asl-alphabet



MILESTONE 2: DATA PREPARATION

Installing necessary Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
import skimage
from PIL import Image
#from skimage.transform import resize
import tensorflow as tf
from tensorflow import keras
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dense, Flatten
from tensorflow.keras.callbacks import EarlyStopping
from keras.models import load_model
```

\

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 stores the data into X and Y

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

train_len = 87000
train_dir = 'C:\\Users\\shala\\Downloads\\asl recognistion\\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=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
```

```python
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
```

```python
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
```

Data Augmentation:
This code generates image data generator which and changes the images into array form
The generators take the image path and label information from data frames and convert them into images. This is done using Scikit Learn IMages using which we first change the size of the image and then they are converted into an array and then stored into X and the label is stored into y. It is then broken into training and testing sets. Then the test data is converted to a binary class matrix using categorical function from the keras.utils library.

```
            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)
```

```
X_data = X_train
y_data = y_train
```

```
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)
```

```
y_cat_train = to_categorical(y_train,29)
y_cat_test = to_categorical(y_test,29)
```

MILESTONE 4 : MODEL BUILDING

```python
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 (MaxPooling2  (None, 30, 30, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 28, 28, 64)        18496

 activation_1 (Activation)   (None, 28, 28, 64)        0

 max_pooling2d_1 (MaxPoolin  (None, 14, 14, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 12, 12, 64)        36928

 activation_2 (Activation)   (None, 12, 12, 64)        0

 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 64)          0
 g2D)

...
Total params: 356637 (1.36 MB)
Trainable params: 356637 (1.36 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
early_stop = EarlyStopping(monitor='val_loss',patience=2)
```

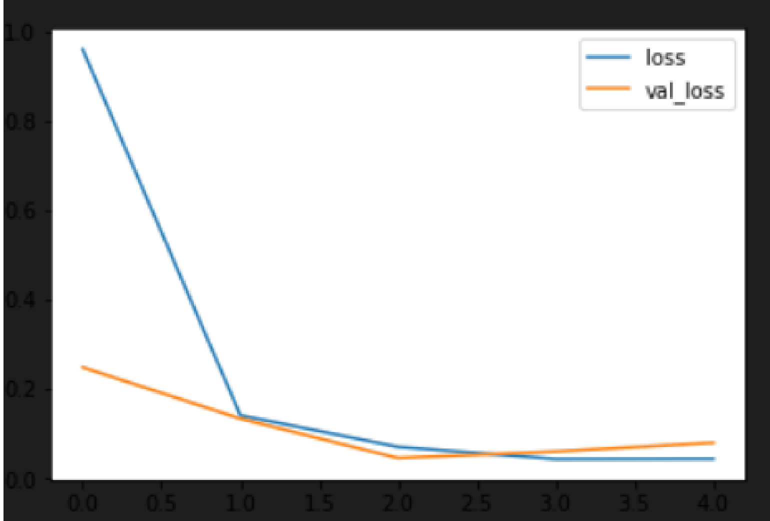
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

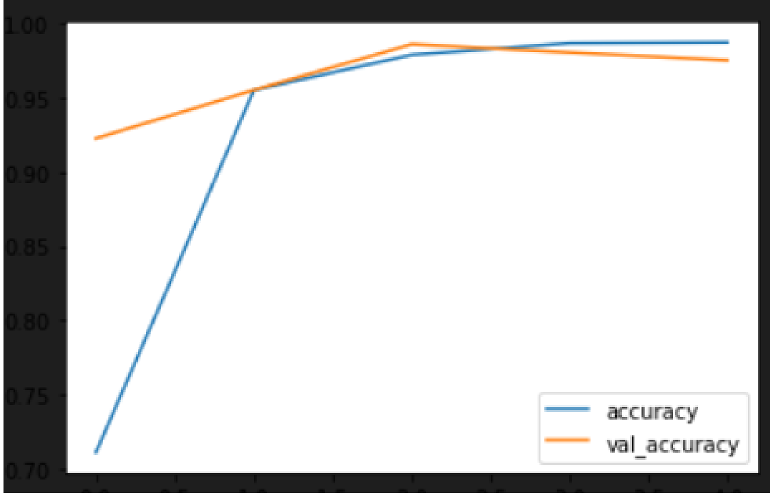
```
model.fit(X_train, y_cat_train,
          epochs=50,
          batch_size=64,
          verbose=2,
          validation_data=(X_test, y_cat_test),
         callbacks=[early_stop])
```

```
Epoch 1/50
952/952 - 116s - loss: 0.9596 - accuracy: 0.7111 - val_loss: 0.2474 - val_accuracy: 0.9228 - 116s/epoch - 122ms/step
Epoch 2/50
952/952 - 105s - loss: 0.1399 - accuracy: 0.9552 - val_loss: 0.1325 - val_accuracy: 0.9553 - 105s/epoch - 110ms/step
Epoch 3/50
952/952 - 105s - loss: 0.0692 - accuracy: 0.9791 - val_loss: 0.0440 - val_accuracy: 0.9862 - 105s/epoch - 110ms/step
Epoch 4/50
952/952 - 106s - loss: 0.0417 - accuracy: 0.9869 - val_loss: 0.0588 - val_accuracy: 0.9807 - 106s/epoch - 111ms/step
Epoch 5/50
952/952 - 104s - loss: 0.0424 - accuracy: 0.9875 - val_loss: 0.0787 - val_accuracy: 0.9754 - 104s/epoch - 109ms/step
```

MILESTONE 5: MODEL EVALUATION

```
metrics = pd.DataFrame(model.history.history)
metrics[['loss','val_loss']].plot()
plt.show()
metrics[['accuracy','val_accuracy']].plot()
plt.show()
```




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

```
[0.07869397103786469, 0.9754406213760376]
```

```python
predictions = model.predictions = np.argmax(model.predict(X_test),axis=1)
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test,predictions))
```

```
816/816 [==============================] - 10s 12ms/step
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       900
           1       1.00      0.98      0.99       900
           2       1.00      1.00      1.00       900
           3       0.94      1.00      0.97       900
           4       0.96      0.97      0.96       900
           5       0.99      0.98      0.99       900
           6       0.98      1.00      0.99       900
           7       0.98      1.00      0.99       900
           8       0.99      0.99      0.99       900
           9       0.99      1.00      1.00       900
          10       0.87      1.00      0.93       900
          11       1.00      0.98      0.99       900
          12       0.98      0.97      0.98       900
          13       0.98      0.99      0.99       900
          14       0.96      0.97      0.97       900
          15       0.98      0.96      0.97       900
          16       0.98      0.99      0.99       900
          17       1.00      0.92      0.96       900
          18       0.99      0.97      0.98       900
          19       0.99      0.97      0.98       900
          20       0.91      1.00      0.95       900
          21       0.98      0.88      0.93       900
...
    accuracy                           0.98     26100
   macro avg       0.98      0.98      0.98     26100
weighted avg       0.98      0.98      0.98     26100
```

Save the model

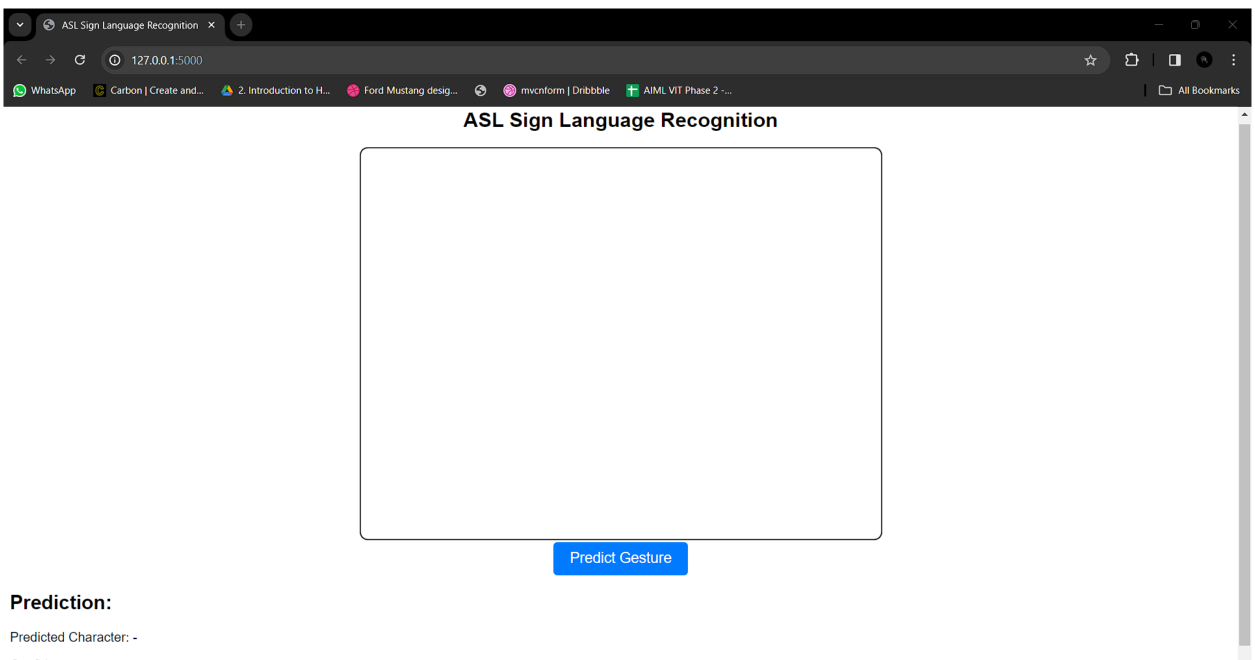
```python
model.save('ASL.h5')
print("Model saved successfully...")
```

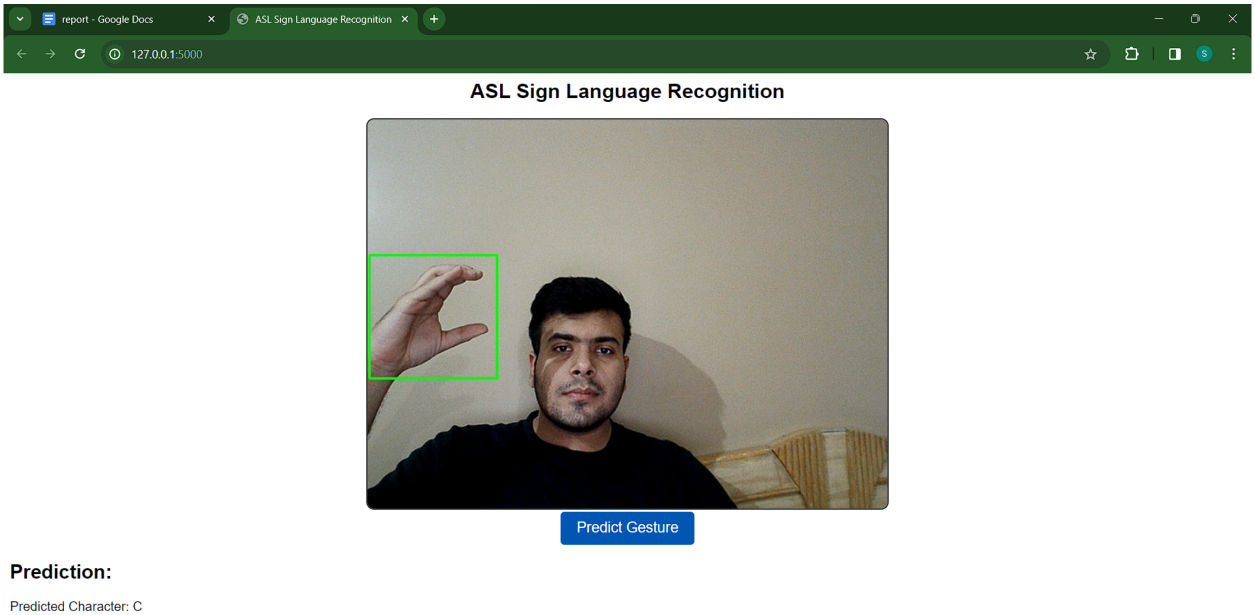
```
C:\Users\shala\anaconda3\lib\site-packages\kera
  saving_api.save_model(
Model saved successfully...
```

Milestone 7: Application Building

Now that we have trained our model, let us build our flask application which will be running in our local browser with a user interface.
In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and places his gesture into the video frame and selects the 'Capture' button, the predicted character is shown below

Activity 1: Create HTML Pages
o We use HTML to create the front end part of the web page.
o  We have created one HTML page index.html and we style it using CSS
o index.html displays the home page and contains the video frame and predictions
For more information regarding HTML
https://www.w3schools.com/html/
o We also use JavaScript-main.js and CSS-main.css to enhance our functionality
and view of HTML pages.






index.html > html > body > div#video-container > button#predict-button

```html
<!DOCTYPE html>
<html>
<head>
    <title>ASL Sign Language Recognition</title>
    <style>
        body {
            text-align: center;
            font-family: Arial, sans-serif;
        }

        h1 {
            font-size: 25px;
            margin-top: 1px;
        }

        #video-container {
            display: flex;
            flex-direction: column;
            align-items: center;
        }

        #video-stream {
            width: 640px;
            height: 480px;
            margin: 2px auto;
            border: 2px solid #333;
            border-radius: 10px;
        }

        #predict-button {
            background-color: #007BFF;
            color: #fff;
            font-size: 18px;
            padding: 10px 20px;
            border: none;
            border-radius: 5px;
            cursor: pointer;
```

```
            border-radius: 5px;
            cursor: pointer;
        }

        #predict-button:hover {
            background-color: #0056b3;
        }

        #prediction-result {
            margin-top: 10px;
            text-align: left;
        }

        #predicted-character {
            font-size: 16px;
        }

        #confidence {
            font-size: 16px;
        }
    </style>
</head>
<body>
    <h1>ASL Sign Language Recognition</h1>
    <div id="video-container">
        <img id="video-stream" src="{{ url_for('index') }}">
        <button id="predict-button" onclick="predictGesturZe()">Predict Gesture</button>
    </div>
    <div id="prediction-result">
        <h2>Prediction:</h2>
        <p id="predicted-character">Predicted Character: -</p>
        <p id="confidence">Confidence: -</p>
```


```
    </style>
</head>
<body>
    <h1>ASL Sign Language Recognition</h1>
    <div id="video-container">
        <img id="video-stream" src="{{ url_for('index') }}">
        <button id="predict-button" onclick="predictGesturZe()">Predict Gesture</button>
    </div>
    <div id="prediction-result">
        <h2>Prediction:</h2>
        <p id="predicted-character">Predicted Character: -</p>
        <p id="confidence">Confidence: -</p>
```


```
    </div>
    <script>
        function predictGesture() {
            fetch('/predict')
                .then(response => response.json())
                .then(data => {
                    document.getElementById('predicted-character').textContent = 'Predicted Character: ' + data.character;
                    document.getElementById('confidence').textContent = 'Confidence: ' + data.confidence + '%';
                });
        }
    </script>
</body>
</html>
```

This is a Python script for a Flask web application that loads a pre-trained deep learning model for
image classification and makes predictions on images uploaded by the user. The app has several
routes, such as the home page ('/'), the prediction page ('/predict.html'). The video feed is displayed and the gesture is recognized and then the image is loaded, preprocessed, and passed through the model for prediction. The predicted result is then displayed on the prediction page. The app can be run by executing the script, and it will start a local server accessible through a web browser.

app.py > predict

```python
from flask import Flask, render_template, Response, request, jsonify,redirect
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import tensorflow as tf
import cv2
import mediapipe as mp
from keras.models import load_model
import numpy as np
import time
import pandas as pd

app = Flask(__name__)
model = load_model('ML/ASL.h5')
cap = cv2.VideoCapture(0)
mphands = mp.solutions.hands
hands = mphands.Hands()
mp_drawing = mp.solutions.drawing_utils

def generate_frames():
    while True:
        success, frame = cap.read()
        if not success:
            break
        h, w ,c= frame.shape
        # w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        # h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

        framergb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        result = hands.process(framergb)
        hand_landmarks = result.multi_hand_landmarks
        if hand_landmarks:
            for handLMs in hand_landmarks:
                x_max = 0
                y_max = 0
                x_min = w
                y_min = h
```

```python
                for lm in handLMs.landmark:
                    x, y = int(lm.x * w), int(lm.y * h)
                    if x > x_max:
                        x_max = x
                    if x < x_min:
                        x_min = x
                    if y > y_max:
                        y_max = y
                    if y < y_min:
                        y_min = y
                y_min -= 20
                y_max += 20
                x_min -= 20
                x_max += 20
                cv2.rectangle(frame, (x_min, y_min), (x_max, y_max), (0, 255, 0), 2)
        ret, buffer = cv2.imencode('.jpg', frame)
        if not ret:
            break
        frame = buffer.tobytes()
        yield (b'--frame\r\n' b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
    cap.release()
    cv2.destroyAllWindows()

@app.route('/')
def home():
    return render_template('index.html',prediction_text='')

@app.route('/videofeed')
def index():
    return Response(generate_frames(), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/predict')
def predict():
    success, frame = cap.read()
    if not success:
        return jsonify({"character": "Error", "confidence": 0})
```

```python
h, w ,c= frame.shape

# w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
# h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
img_counter = 0
analysisframe = ''
letterpred = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y']

print("Space hit, capturing...")
analysisframe = frame
framergbanalysis = cv2.cvtColor(analysisframe, cv2.COLOR_BGR2RGB)
resultanalysis = hands.process(framergbanalysis)
hand_landmarksanalysis = resultanalysis.multi_hand_landmarks
if hand_landmarksanalysis:
    for handLMsanalysis in hand_landmarksanalysis:
        x_max = 0
        y_max = 0
        x_min = w
        y_min = h
        for lmanalysis in handLMsanalysis.landmark:
            x, y = int(lmanalysis.x * w), int(lmanalysis.y * h)
            if x > x_max:
                x_max = x
            if x < x_min:
                x_min = x
            if y > y_max:
                y_max = y
            if y < y_min:
                y_min = y
        y_min -= 20
        y_max += 20
        x_min -= 20
        x_max += 20

```

```python
    analysisframe = cv2.cvtColor(analysisframe, cv2.COLOR_BGR2GRAY)
    analysisframe = analysisframe[y_min:y_max, x_min:x_max]
    analysisframe = cv2.resize(analysisframe,(64,64))
    nlist = []
    rows,cols = analysisframe.shape
    for i in range(rows):
        for j in range(cols):
            k = analysisframe[i,j]
            nlist.append(k)

    datan = pd.DataFrame(nlist).T
    colname = []
    for val in range(4096):
        colname.append(val)
    datan.columns = colname

    pixeldata = datan.values
    pixeldata = pixeldata/ 255
    pixeldata = pixeldata.reshape(-1,64,64,1)
    prediction = model.predict(pixeldata)
    predarray = np.array(prediction[0])
    letter_prediction_dict = {letterpred[i]: predarray[i] for i in range(len(letterpred))}
    predarrayordered = sorted(predarray, reverse=True)
    high1 = predarrayordered[0]
    high2 = predarrayordered[1]
    high3 = predarrayordered[2]
    for key,value in letter_prediction_dict.items():
        if value==high1:
            print("Predicted Character 1: ", key)
            print('Confidence 1: ', 100*value)
            character = key
            confidence = 100 * value
            return jsonify({"character": character, "confidence": confidence})
        elif value==high2:
            print("Predicted Character 2: ", key)
            print('Confidence 2: ', 100*value)
            character = key
            confidence = 100 * value
            return jsonify({"character": character, "confidence": confidence})
        elif value==high3:
            print("Predicted Character 3: ", key)
            print('Confidence 3: ', 100*value)
    return redirect("/")


if __name__ == '__main__':
    app.run(debug=True)
```

To run this Flask application, simply navigate to the project directory in the terminal and run the command "python app.py". This will start the Flask server, and you can access the web application by

visiting the local host address in your web browser. Once you upload an image and submit the form,

the application will use the trained model to predict the species of the plant in the image and display
the result on the page.