Project Development Phase
Project Manual

Date	21 November 2023
Team ID	Team- 592184
Project Name	ASL - Alphabet Image Recognition
Maximum Marks	15 Marks

ASL (American Sign Language) - Alphabet Image recognition

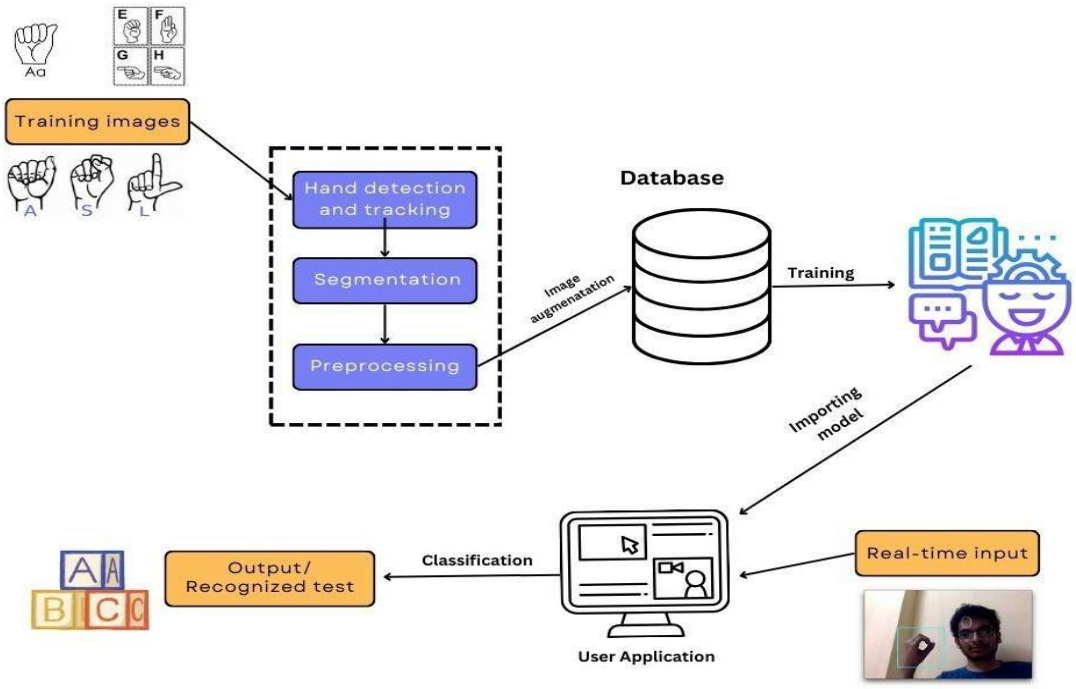
Introduction:-

American Sign Language (ASL) serves as the primary means of communication for the deaf community in North America, employing a visual language that incorporates hand gestures, facial expressions, and body movements to convey meaning. In recent times, there has been a notable surge of interest in developing technologies to bridge the communication gap between the deaf and hearing communities. One significant technological advancement is the ASL Alphabet Image Recognition, a machine learning-based image classification task designed to identify the 26 letters of the English alphabet.

This project goes beyond mere letter recognition, also encompassing three additional classes for recognizing signs representing "space," "delete," and "nothing." The central objective of ASL Alphabet Image Recognition is to train a machine learning model capable of classifying images depicting hand signs corresponding to each letter of the alphabet. The potential application of this trained model extends to integration into real-time recognition applications for ASL alphabet signs from video streams. Such applications could play a crucial role in enhancing communication between the deaf and hearing communities, providing a practical solution to facilitate understanding and interaction across various contexts.

The ASL Alphabet Image Recognition project represents a significant technological stride towards fostering inclusivity and accessibility. In essence, it is an initiative aimed at easing communication between deaf and hearing individuals. Through the utilization of machine learning for the classification of ASL hand signs, the project lays the groundwork for the development of real-time applications that could revolutionize interaction dynamics. Ultimately, these technological advancements have the potential to contribute to a more inclusive and connected society by dismantling communication barriers between different linguistic communities.

Technical Architecture:-



Prerequisites:

Deep Learning Concepts
Convolutional neural network
Flask rewrite

Project Objectives:

1. Gain a firm understanding of the core principles and methodologies linked to Convolutional Neural Networks.
2. Cultivate an in-depth comprehension of image data and its diverse applications.
3. Acquire expertise in preprocessing and cleansing data using a variety of data preprocessing techniques.
4. Develop the knowledge and competence necessary to build a web application utilizing the Flask framework.

Project Flow:

1. User Interaction: Interact with the User Interface (UI) for image selection.

2. Model Integration: The selected image is analyzed by the integrated model within the

Flask application.

3. CNN Model Analysis: Convolutional Neural Network (CNN) models assess the image, and predictions are presented on the Flask UI.

To fulfill these goals, the following tasks must be accomplished:

- Gather Data.

- Preprocess Data.

- Build the Model.

- Train the Model.

- Evaluate the Model.

- Deploy the Model.

PROJECT DEVELOPMENT:-

DATA COLLECTION:-

The Dataset used in this project is collected from the following link:
https://www.kaggle.com/datasets/grassknoted/asl-alphabet
And it is used in our project with Kaggle API credentials in the following way-

Kaggle API credentials:-

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

Download and unzip dataset:-

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

```
Downloading asl-alphabet.zip to /content
 99% 1.02G/1.03G [00:10<00:00, 166MB/s]
100% 1.03G/1.03G [00:10<00:00, 105MB/s]
```

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

DATA PREPARATION:-

Importing the necessary libraries:-

```
# Load Data
import os
import cv2
import numpy as np

# Data Visualisation
import matplotlib.pyplot as plt

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

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

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

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

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

Configuring the parameters:-

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

Creating labels:-

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

```
['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', 's
```

DATA PREPROCESSING:-

Creating metadata:-

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

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

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

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

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

```
[0] def data_augmentation():
```

DATA AUGMENTATION:-

Applying data augmentation to train, test, validation data:-

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

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

    return train_generator, validation_generator, test_generator

# Seed for reproducibility
seed_everything(2253)

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

```
Found 69600 validated image filenames belonging to 29 classes.
Found 19886 validated image filenames belonging to 29 classes.
Found 17400 validated image filenames belonging to 29 classes.
```

MODEL BUILDING:-

We are using VGG16 Model and the weights have been taken from ImageNet Model

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

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

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

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

# Summarize the model architecture
model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.
58889256/58889256 [==============================] - 0s 0us/step
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 block1_conv1 (Conv2D)       (None, 32, 32, 64)        1792

 block1_conv2 (Conv2D)       (None, 32, 32, 64)        36928

 block1_pool (MaxPooling2D)  (None, 16, 16, 64)        0

 block2_conv1 (Conv2D)       (None, 16, 16, 128)       73856

 block2_conv2 (Conv2D)       (None, 16, 16, 128)       147584

 block2_pool (MaxPooling2D)  (None, 8, 8, 128)         0

 block3_conv1 (Conv2D)       (None, 8, 8, 256)         295168

 block3_conv2 (Conv2D)       (None, 8, 8, 256)         590080

 block3_conv3 (Conv2D)       (None, 8, 8, 256)         590080

 block3_pool (MaxPooling2D)  (None, 4, 4, 256)         0

 block4_conv1 (Conv2D)       (None, 4, 4, 512)         1180160

 block4_conv2 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_conv3 (Conv2D)       (None, 4, 4, 512)         2359808

 block4_pool (MaxPooling2D)  (None, 2, 2, 512)         0

 block5_conv1 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv2 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_conv3 (Conv2D)       (None, 2, 2, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 1, 1, 512)         0

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 128)               65664

 dense_1 (Dense)             (None, 29)                3741

=================================================================
Total params: 14784093 (56.40 MB)
Trainable params: 14784093 (56.40 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compiling and training the model:-

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

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

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
Epoch 1/10
543/543 [==============================] - ETA: 0s - loss: 2.5409 - accuracy: 0.1969
Epoch 1: val_accuracy improved from -inf to 0.42445, saving model to /content/sample_data/best_model_weights.h5
543/543 [==============================] - 163s 269ms/step - loss: 2.5409 - accuracy: 0.1969 - val_loss: 1.7111 - val_accuracy: 0.4244
Epoch 2/10
543/543 [==============================] - ETA: 0s - loss: 1.1009 - accuracy: 0.6191
Epoch 2: val_accuracy improved from 0.42445 to 0.73911, saving model to /content/sample_data/best_model_weights.h5
543/543 [==============================] - 142s 261ms/step - loss: 1.1009 - accuracy: 0.6191 - val_loss: 0.7840 - val_accuracy: 0.7391
Epoch 3/10
543/543 [==============================] - ETA: 0s - loss: 0.5650 - accuracy: 0.8141
Epoch 3: val_accuracy improved from 0.73911 to 0.87142, saving model to /content/sample_data/best_model_weights.h5
543/543 [==============================] - 125s 230ms/step - loss: 0.5650 - accuracy: 0.8141 - val_loss: 0.4190 - val_accuracy: 0.8714
Epoch 4/10
543/543 [==============================] - ETA: 0s - loss: 0.3615 - accuracy: 0.8875
Epoch 4: val_accuracy did not improve from 0.87142
543/543 [==============================] - 143s 263ms/step - loss: 0.3615 - accuracy: 0.8875 - val_loss: 0.4847 - val_accuracy: 0.8604
Epoch 5/10
543/543 [==============================] - ETA: 0s - loss: 0.3016 - accuracy: 0.9099
Epoch 5: val_accuracy improved from 0.87142 to 0.93473, saving model to /content/sample_data/best_model_weights.h5
543/543 [==============================] - 125s 230ms/step - loss: 0.3016 - accuracy: 0.9099 - val_loss: 0.2324 - val_accuracy: 0.9347
Epoch 6/10
543/543 [==============================] - ETA: 0s - loss: 0.2637 - accuracy: 0.9230
Epoch 6: val_accuracy did not improve from 0.93473
543/543 [==============================] - 143s 263ms/step - loss: 0.2637 - accuracy: 0.9230 - val_loss: 0.2255 - val_accuracy: 0.9334
Epoch 7/10
543/543 [==============================] - ETA: 0s - loss: 0.2280 - accuracy: 0.9352
Epoch 7: val_accuracy improved from 0.93473 to 0.94279, saving model to /content/sample_data/best_model_weights.h5
543/543 [==============================] - 147s 270ms/step - loss: 0.2280 - accuracy: 0.9352 - val_loss: 0.2000 - val_accuracy: 0.9428
Epoch 8/10
543/543 [==============================] - ETA: 0s - loss: 0.2205 - accuracy: 0.9377
Epoch 8: val_accuracy did not improve from 0.94279
543/543 [==============================] - 129s 237ms/step - loss: 0.2205 - accuracy: 0.9377 - val_loss: 0.2174 - val_accuracy: 0.9416
Epoch 9/10
543/543 [==============================] - ETA: 0s - loss: 0.2054 - accuracy: 0.9439
Epoch 9: val_accuracy improved from 0.94279 to 0.95756, saving model to /content/sample_data/best_model_weights.h5
543/543 [==============================] - 127s 234ms/step - loss: 0.2054 - accuracy: 0.9439 - val_loss: 0.1446 - val_accuracy: 0.9576
Epoch 10/10
543/543 [==============================] - ETA: 0s - loss: 0.1876 - accuracy: 0.9472
Epoch 10: val_accuracy did not improve from 0.95756
543/543 [==============================] - 128s 235ms/step - loss: 0.1876 - accuracy: 0.9472 - val_loss: 0.1809 - val_accuracy: 0.9490
```

MODEL EVALUATION:-

Evaluating the model using accuracy, confusion matrix and giving classification report:-

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

```
136/136 [==============================] - 23s 169ms/step - loss: 0.1744 - accuracy: 0.9503
Evaluate Test Accuracy: 95.034480%
```

```
# Confusion Matrix:-
fine_tuned_model = load_model("/content/sample_data/best_model_weights.h5")
predictions = fine_tuned_model.predict(test_generator)
# Get the true labels from the generator
true_labels = test_generator.classes
# Compute the confusion matrix using tf.math.confusion_matrix
confusion_matrix = tf.math.confusion_matrix(
labels = true_labels,
predictions = predictions.argmax(axis=1),
num_classes = 29
)
```

```
136/136 [==============================] - 24s 171ms/step
```

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

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

23s

```
136/136 [==============================] - 24s 173ms/step
              precision    recall  f1-score   support

           A       0.97      0.85      0.91       600
           B       0.94      0.97      0.96       600
           C       0.98      0.99      0.99       600
           D       0.99      0.98      0.98       600
           E       0.95      0.95      0.95       600
           F       1.00      0.97      0.98       600
           G       0.93      0.98      0.96       600
           H       0.97      0.98      0.98       600
           I       0.96      0.87      0.91       600
           J       0.90      0.98      0.94       600
           K       0.90      0.93      0.92       600
           L       1.00      0.98      0.99       600
           M       0.82      0.95      0.88       600
           N       0.96      0.91      0.94       600
           O       0.97      0.97      0.97       600
           P       0.99      0.98      0.99       600
           Q       0.98      0.99      0.98       600
           R       0.92      0.81      0.86       600
           S       0.87      0.94      0.90       600
           T       0.96      0.97      0.97       600
           U       0.86      0.94      0.90       600
           V       0.92      0.89      0.91       600
           W       0.99      0.95      0.97       600
           X       0.95      0.88      0.91       600
           Y       0.98      0.98      0.98       600
           Z       0.94      0.97      0.96       600
         del       0.97      0.97      0.97       600
     nothing       0.98      0.99      0.99       600
       space       0.99      0.99      0.99       600

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

LOAD AND TEST THE MODEL:-

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

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

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

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

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

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

```
1/1 [==============================] - 0s 472ms/step
The image is predicted to belong to class: Y
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

APPLICATION BUILDING:-

1. Creating an HTML Page.
2. Adding styles to it using css.
3. Adding actions to it using javascript.
4. Creating App.py python script for web application that uses the model for image classification predictions.
5. Executing these files using Spyder IDE.

HTML CODE:-

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

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

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

    <div class="container-wrapper">
        <div class="info-container">
            <div class="info-section">
                <p>The American Sign Language (ASL) is the primary language used by deaf individuals in North America. It is a visual language that uses
            </div class="image-section" >
            <p> Take a look at American Sign Language through the picture below, it contains the alphabets from A-Z, and additionally 3 symbols for spac
            <img src="https://www.dictionary.com/e/wp-content/uploads/2018/01/american_sign_language4-790x310.jpg" alt="ASL image">
        </div>

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

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

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

            <div id="result"> </div>
        </div>
    </div>



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

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

</html>
```

CSS CODE:-

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

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

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

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

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

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

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

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

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

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

.upload-label {
    margin-top: 10px;
}
```

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

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

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

#result {
    position: absolute;
    bottom:1px;
    width: 100%;
    font-weight: bold;
    text-align: center;
    color: #333;
}
```

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

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

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

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

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

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

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

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

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

#file-input {
    display: none;
}
```

JAVASCRIPT CODE:-

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

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

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

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

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

            generateImages(videoURL);
        }
    });

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

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

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

                const img = new Image();
                img.src = canvas.toDataURL('image/png');
                img.alt = `Frame ${i}`;
                imageContainer.appendChild(img);
            }
            console.log('Images generated successfully.');
        });
    }
});
```

```
function previewImage(input) {
            const preview = document.getElementById('image-preview');
            preview.innerHTML = '';

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

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

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

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

function previewImages(input) {
    console.log('Previewing multiple images.');
    var previewContainer = document.getElementById('image-preview');

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

    var files = input.files;

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

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

        reader.readAsDataURL(file);
    }
}
```

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

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

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

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

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

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

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

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

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

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

    fetch('/predict', {
        method: 'POST',
        body: formData
```

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

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

    const formData = new FormData(form);

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

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

App.py CODE:-

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

app = Flask(__name__)

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

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

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

# Your existing Python code

def predict_image(file_path):
    labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',
              'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', 'del', 'nothing', 'space']

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

    # Make prediction
    prediction = model.predict(img_array)

    predicted_class = labels[np.argmax(prediction)]
    print('Predicted class:', predicted_class)  # Log the predicted class

    return predicted_class

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

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

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


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

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

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

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

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

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

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

            predicted_word = ''.join(predictions)

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

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

WEB APPLICATION:-

Getting confused with the ASL 😳, Use our Deep learning tool to easily predict the ASL sign

Want to predict something?

Single Video | Series of Images

Predict

Prediction: Your image represents P

Getting confused with the ASL 😳, Use our Deep learning tool to easily predict the ASL sign

Want to predict something?

Single Video | Series of Images

Predict

Prediction: Your images represent ANT