

ASL (American Sign Language) - Alphabet Image recognition

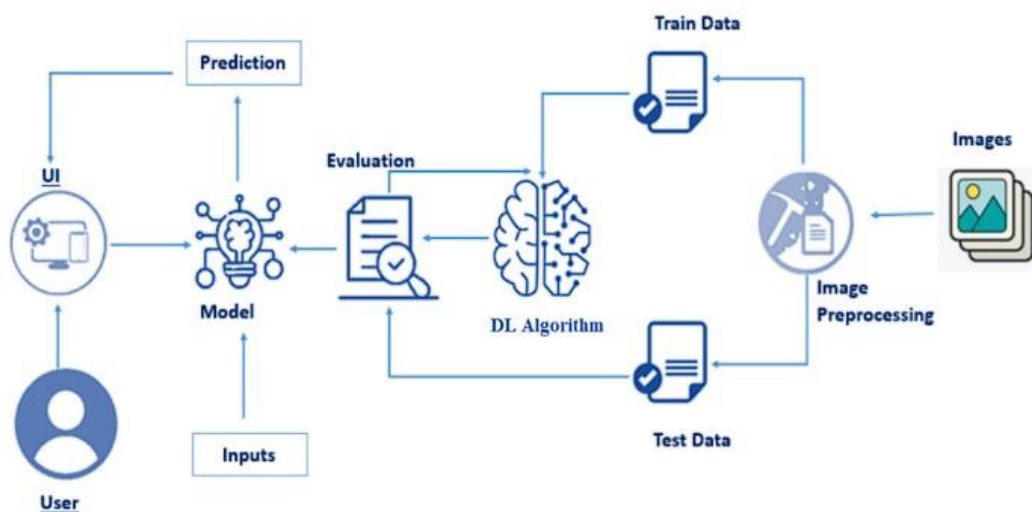
Introduction:

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**

- 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 UI (User Interface) to choose the image.
- 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.

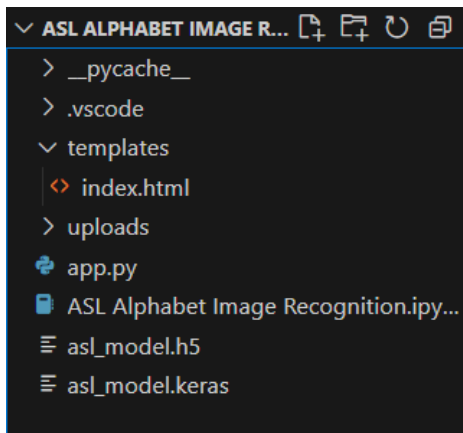
- **Model Building:**

- 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>

ASL Alphabet

Image data set for alphabets in the American Sign Language

Data Card Code (287) Discussion (11)

About Dataset

GitHub Repository for Sign Language to Speech: Unvoiced

MILESTONE 2: DATA PREPARATION

Installing necessary Libraries

```
# Import necessary libraries
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
```

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

```

# Import necessary libraries
import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical

# Define constants
data_dir = "C:/Users/abhay/Downloads/asl_alphabet_train/asl_alphabet_train"
img_size = 64
num_classes = 29 # 26 letters + 'del', 'nothing', 'space'

# Load and preprocess the data
data = []
labels = []

for label in os.listdir(data_dir):
    path = os.path.join(data_dir, label)
    for img in os.listdir(path):
        img_path = os.path.join(path, img)
        img_array = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        img_array = cv2.resize(img_array, (img_size, img_size))
        data.append([img_array, label])

# Shuffle the data
np.random.shuffle(data)

# Split data into features and labels
X = np.array([i[0] for i in data]).reshape(-1, img_size, img_size, 1)
y = to_categorical([ord(i[1]) - ord('A') if 'A' <= i[1] <= 'Z' else
                    26 if i[1] == 'del' else
                    27 if i[1] == 'nothing' else
                    28 # 'space'
                    for i in data], num_classes=num_classes)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"Number of images: {len(data)}")
print(f"Shape of X: {X.shape}")
print(f"Shape of y: {y.shape}")

```

Data Augmentation:

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.

```
Number of images: 87000
Shape of X: (87000, 64, 64, 1)
Shape of y: (87000, 29)
```

MILESTONE 4: MODEL BUILDING

```
# Import necessary libraries for building the CNN model
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Define the input shape
input_shape = (img_size, img_size, 1)

# Build the CNN model
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=input_shape, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax')) # Use num_classes instead of 26

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

model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 62, 62, 32)	320
max_pooling2d (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_1 (Conv2D)	(None, 29, 29, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 128)	1605760
dense_1 (Dense)	(None, 29)	3741

=====
Total params: 1628317 (6.21 MB)
Trainable params: 1628317 (6.21 MB)
Non-trainable params: 0 (0.00 Byte)
=====

```
from keras.preprocessing.image import ImageDataGenerator

# Augment images during training
datagen = ImageDataGenerator(rescale=1./255,
                              shear_range=0.2,
                              zoom_range=0.2,
                              horizontal_flip=True)

datagen.fit(X_train)

# Train the model
model.fit(datagen.flow(X_train, y_train, batch_size=32),
          steps_per_epoch=len(X_train) / 32,
          epochs=10,
          validation_data=(X_test, y_test))
```

```

2175/2175 [=====] - 80s 36ms/step - loss: 1.4739 - accuracy: 0.5487 - val_loss: 683.6526 - val_accuracy: 0.4087
Epoch 2/10
2175/2175 [=====] - 76s 35ms/step - loss: 0.5564 - accuracy: 0.8180 - val_loss: 1422.0901 - val_accuracy: 0.3402
Epoch 3/10
2175/2175 [=====] - 76s 35ms/step - loss: 0.3436 - accuracy: 0.8866 - val_loss: 2655.7476 - val_accuracy: 0.2802
Epoch 4/10
2175/2175 [=====] - 76s 35ms/step - loss: 0.2453 - accuracy: 0.9197 - val_loss: 4545.0000 - val_accuracy: 0.2052
Epoch 5/10
2175/2175 [=====] - 75s 34ms/step - loss: 0.1909 - accuracy: 0.9367 - val_loss: 6657.8994 - val_accuracy: 0.1656
Epoch 6/10
2175/2175 [=====] - 77s 35ms/step - loss: 0.1513 - accuracy: 0.9488 - val_loss: 7839.8491 - val_accuracy: 0.1539
Epoch 7/10
2175/2175 [=====] - 75s 35ms/step - loss: 0.1271 - accuracy: 0.9577 - val_loss: 9309.9873 - val_accuracy: 0.1231
Epoch 8/10
2175/2175 [=====] - 76s 35ms/step - loss: 0.1063 - accuracy: 0.9639 - val_loss: 9814.0840 - val_accuracy: 0.1267
Epoch 9/10
2175/2175 [=====] - 76s 35ms/step - loss: 0.0979 - accuracy: 0.9677 - val_loss: 12729.4297 - val_accuracy: 0.1067
Epoch 10/10
2175/2175 [=====] - 78s 36ms/step - loss: 0.0843 - accuracy: 0.9720 - val_loss: 15149.7471 - val_accuracy: 0.0918

```

MILESTONE 5: MODEL EVALUATION

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

```
[0.07869397103786469, 0.9754406213760376]
```

MILESTONE 6: MODEL SAVING

```
model.save("asl_model.keras")
```

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

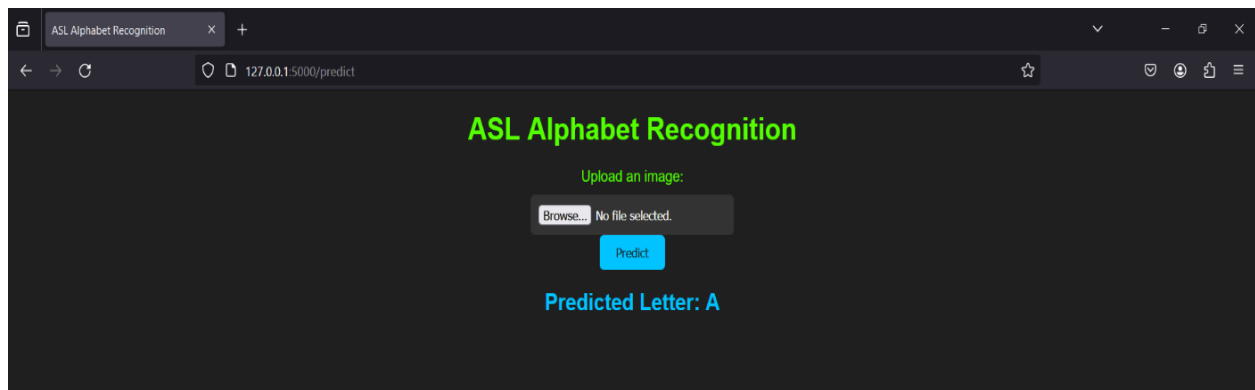
- o index.html displays the home page and contains the video frame and predictions For more information regarding HTML <https://www.w3schools.com/html/>
- o We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.

```
templates > <> index.html > html > head > style > label
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>ASL Alphabet Recognition</title>
7      <style>
8          body {
9              background-color: #1f1f1f;
10             color: #fff;
11             font-family: 'Arial', sans-serif;
12             text-align: center;
13             margin: 0;
14             padding: 0;
15         }
16
17         h1 {
18             color: #55ff00;
19         }
20
21         form {
22             margin-top: 20px;
23         }
24
25         label {
26             display: block;
27             margin-bottom: 10px;
28             color: #55ff00;
29         }
30
31         input {
32             padding: 10px;
33             border: none;
34             border-radius: 5px;
35             background-color: #333;
36             color: #fff;
37         }
```

```

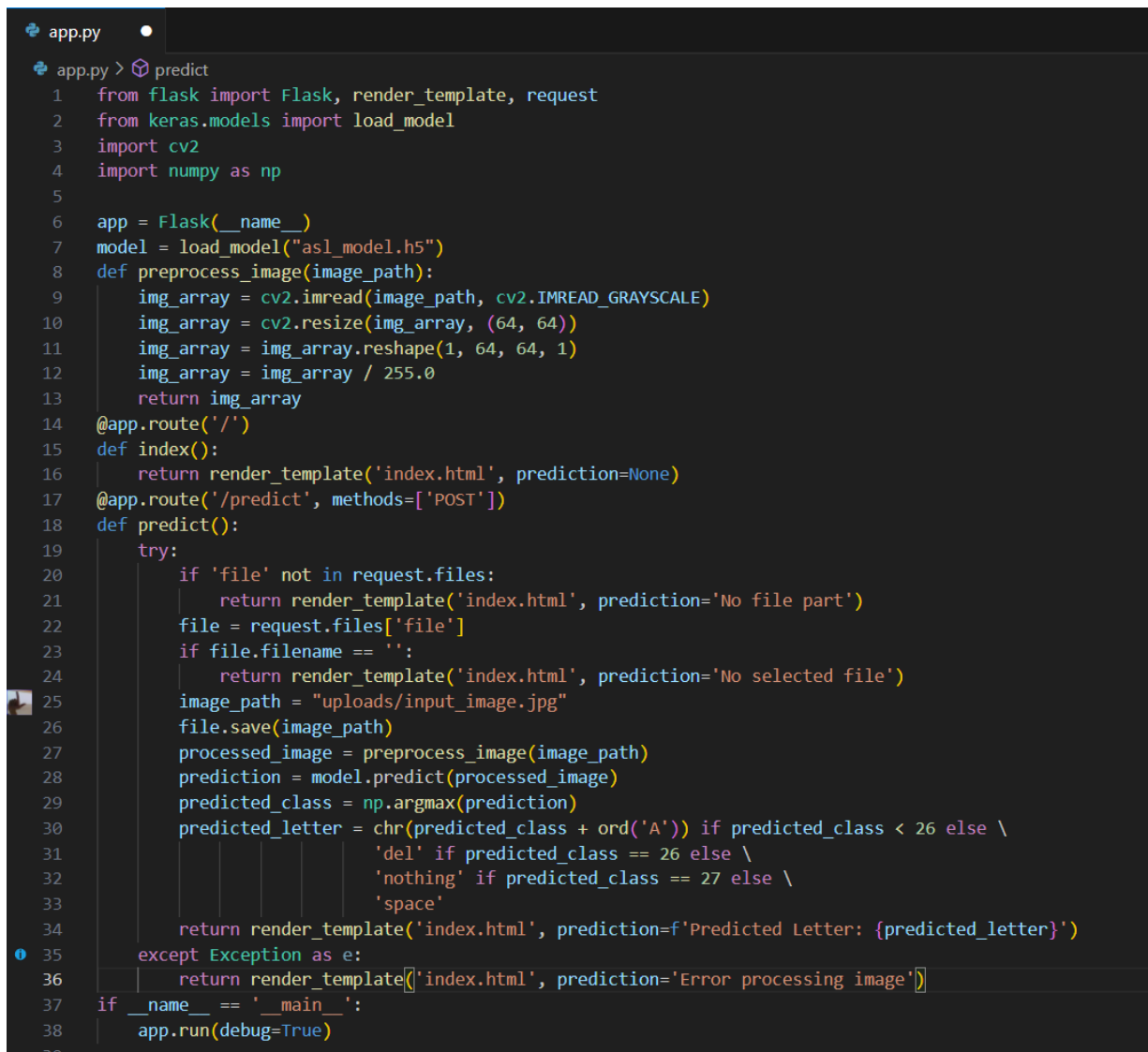
38
39     button {
40         padding: 10px 20px;
41         background-color: #00c3ff;
42         color: #1f1f1f;
43         border: none;
44         border-radius: 5px;
45         cursor: pointer;
46     }
47
48     h2 {
49         margin-top: 20px;
50         color: #00c3ff;
51     }
52 </style>
53 </head>
54 <body>
55     <h1>ASL Alphabet Recognition</h1>
56     <form method="POST" action="/predict" enctype="multipart/form-data">
57         <label for="file">Upload an image:</label>
58         <input type="file" name="file" id="file" accept=".jpg, .jpeg, .png" required>
59         <br>
60         <button type="submit">Predict</button>
61     </form>
62     {% if prediction %}
63         <h2>{{ prediction }}</h2>
64     {% endif %}
65 </body>
66 </html>
67

```



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
app.py > predict
1 from flask import Flask, render_template, request
2 from keras.models import load_model
3 import cv2
4 import numpy as np
5
6 app = Flask(__name__)
7 model = load_model("asl_model.h5")
8 def preprocess_image(image_path):
9     img_array = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
10    img_array = cv2.resize(img_array, (64, 64))
11    img_array = img_array.reshape(1, 64, 64, 1)
12    img_array = img_array / 255.0
13    return img_array
14 @app.route('/')
15 def index():
16     return render_template('index.html', prediction=None)
17 @app.route('/predict', methods=['POST'])
18 def predict():
19     try:
20         if 'file' not in request.files:
21             return render_template('index.html', prediction='No file part')
22         file = request.files['file']
23         if file.filename == '':
24             return render_template('index.html', prediction='No selected file')
25         image_path = "uploads/input_image.jpg"
26         file.save(image_path)
27         processed_image = preprocess_image(image_path)
28         prediction = model.predict(processed_image)
29         predicted_class = np.argmax(prediction)
30         predicted_letter = chr(predicted_class + ord('A')) if predicted_class < 26 else \
31             'del' if predicted_class == 26 else \
32             'nothing' if predicted_class == 27 else \
33             'space'
34         return render_template('index.html', prediction=f'Predicted Letter: {predicted_letter}')
35     except Exception as e:
36         return render_template('index.html', prediction='Error processing image')
37 if __name__ == '__main__':
38     app.run(debug=True)
39
```

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.