**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, we must the following software’s, concepts, and packages:

* Software’s
  + Google colab
  + VS code
  + Spyder IDE.
* Deep Learning Concepts
  + CNN: A Convolutional Neural Network is a class of deep neural networks, most commonly applied to analysing visual imagery. For this project, we have used DenseNet architecture.
  + Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.
* Packages
  + TensorFlow
  + Keras

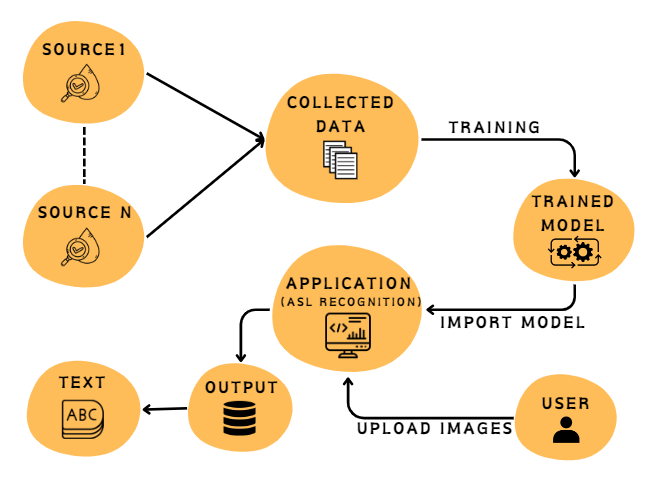
**Project Objectives:**

* Know fundamental concepts and techniques of Convolutional Neural Networks.
* Gain a broad understanding of ASL image data.
* Train and fine tune parameters for the best model accuracy.
* Know how to build a web application using the Flask framework.
* Build a complete website and interface with model to predict the sign language images in real time.

**Project Flow:**

* The user interacts with the UI(User Interface), i.e., our website to choose the image and upload it.
* The chosen image is analysed by the DenseNet model which is integrated with the flask application. After the analysis, the prediction is showcased on the website.

Our solution architecture looks as follows:



Steps involved to execute this are:

* Data Collection: Download the dataset using Google Colab from Kaggle.
* Data Preprocessing: Preprocess the data by rescaling, zooming, etc.
* Model Building:
  + Import the necessary libraries for building the CNN model.
  + Define the input shape of the image data.
  + Initialise the model and add layers to the model:
    - Convolutional Layers: Apply filters to the input image to create feature maps.
    - Fully Connected Layers: Flatten the output of the convolutional layers and apply fully connected layers to classify the images
  + Compile the model using Adam 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.
* Model Deployment: Save the model for future use and deploy it in real-world applications.
* Flask Development: Develop a user interface and integrate it with the saved model using Flask.

**Project Structure:**

Create a Project folder which contains files as shown below 

**1. DATA COLLECTION**

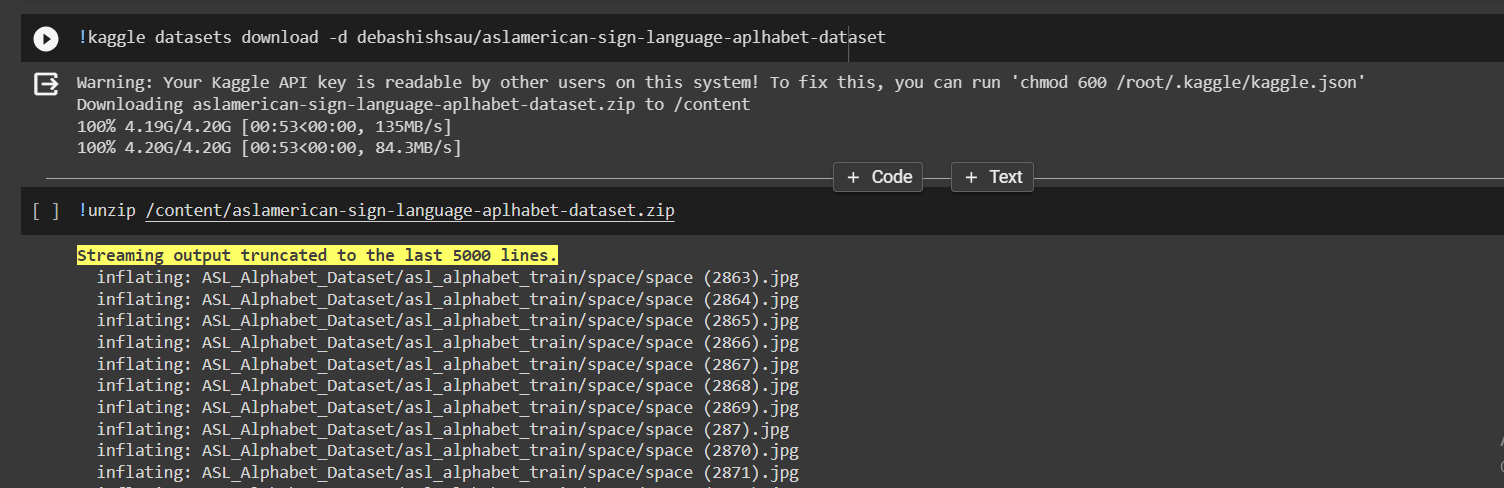
**Kaggle data set link :**

<https://www.kaggle.com/datasets/debashishsau/aslamerican-sign-language-aplhabet-dataset>

Below three lines of code are used to set up the Kaggle API credentials in a Google Colab notebook. The first line installs the Kaggle. The second line creates a directory named .kaggle in the root directory of the Colab notebook. The third line copies the kaggle.json file (which contains the API credentials) to the .kaggle directory.

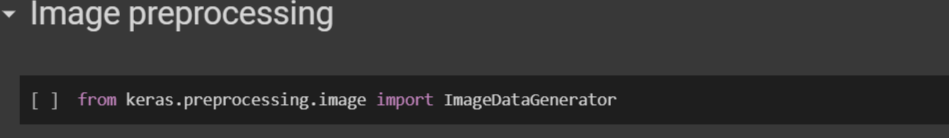


This code downloads the ASL Alphabet dataset from Kaggle.

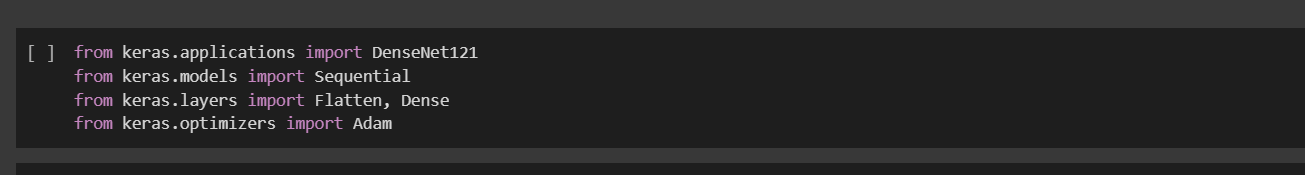


**2. DATA PREPARATION**

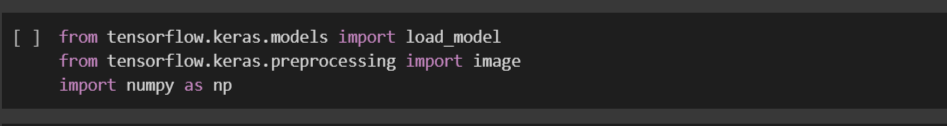
* Installing Imagedata Generator



* Model Building

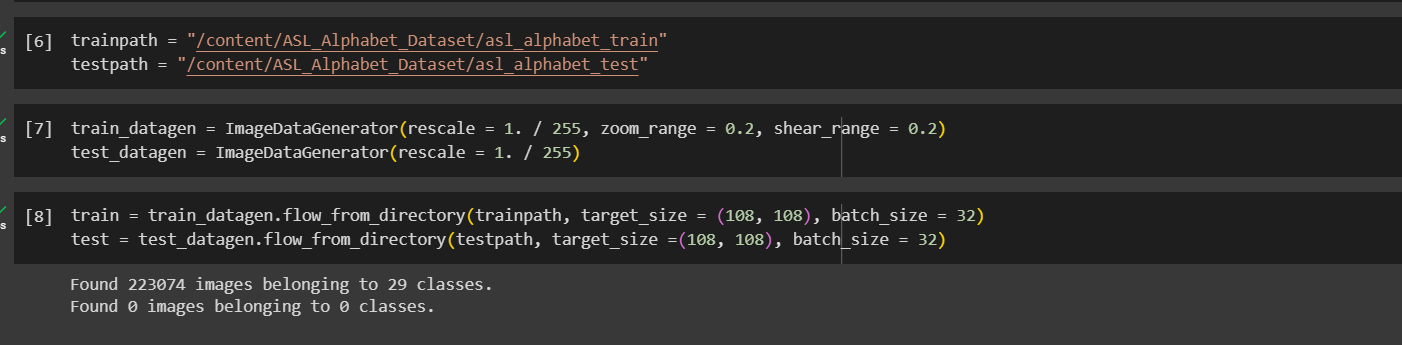


* Test



* Configuration

This snippet of code gives us the path to the train and test folders that are present in the ASL\_Alphabet\_Dataset folder.



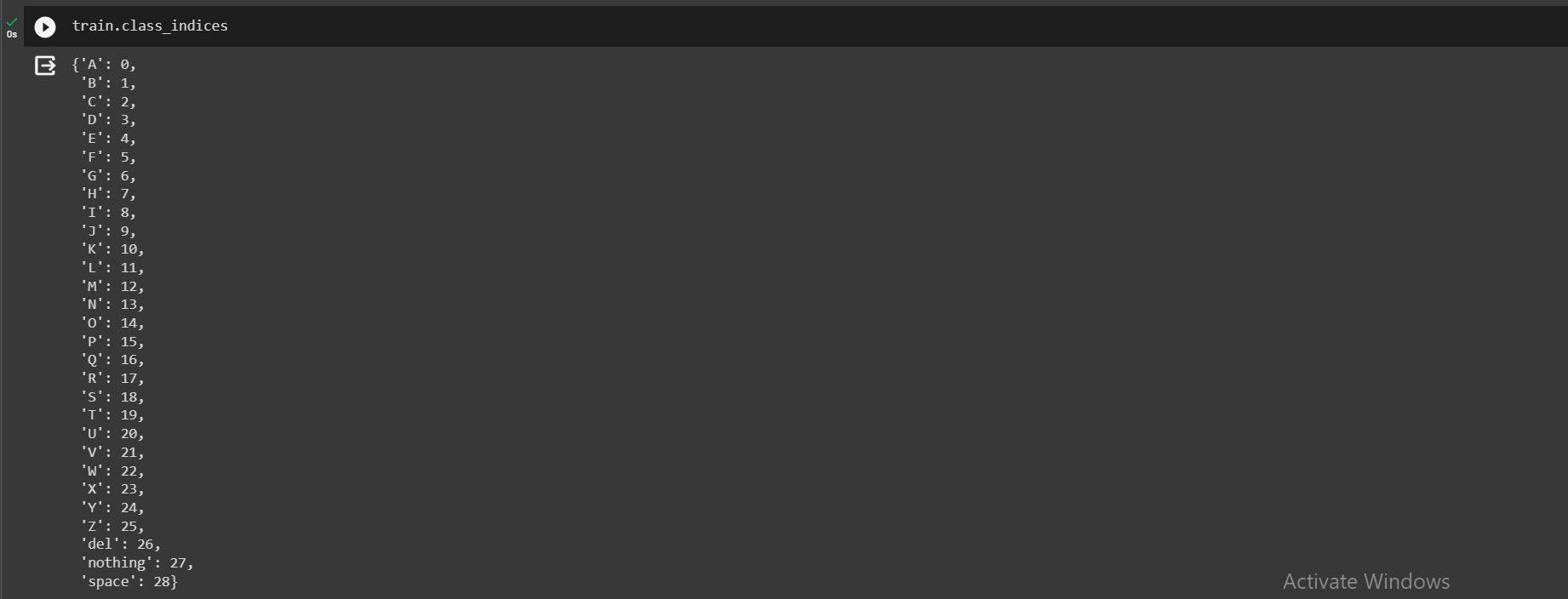
**3. DATA PREPROCESSING**

* The following snippet of code shows the data augmentation. It generates image data generators using the ImageDataGenerator class from the Keras library for the train and test set of images. The generators take the images from the train and test folder and apply the same data augmentation technique of rescaling the image pixel values to a range of 0 to 1.



* Train labels/Indices

This line of code prints the class labels we have in the training and testing data.



**4. MODEL BUILDING**

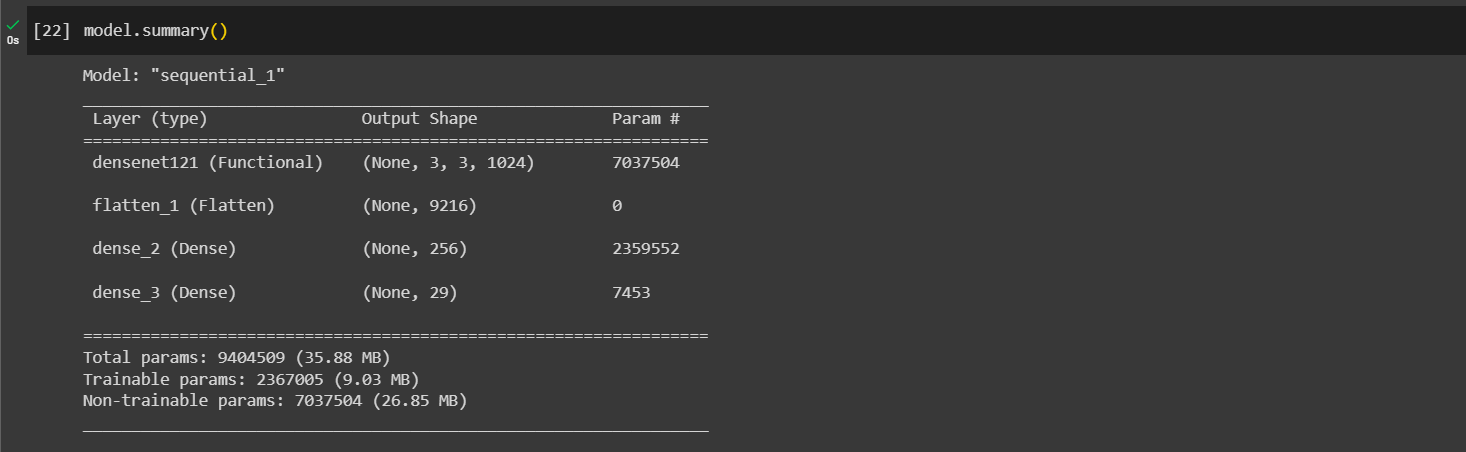
* Load the pre-trained DenseNet model.

****

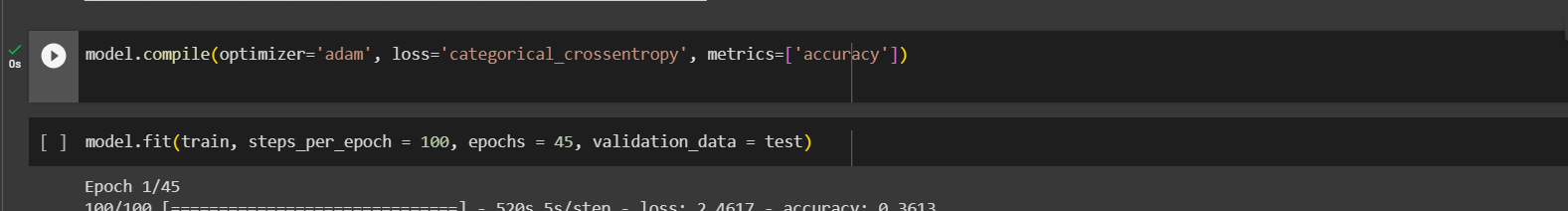
* Initialise the model and add the layers to the model.

****

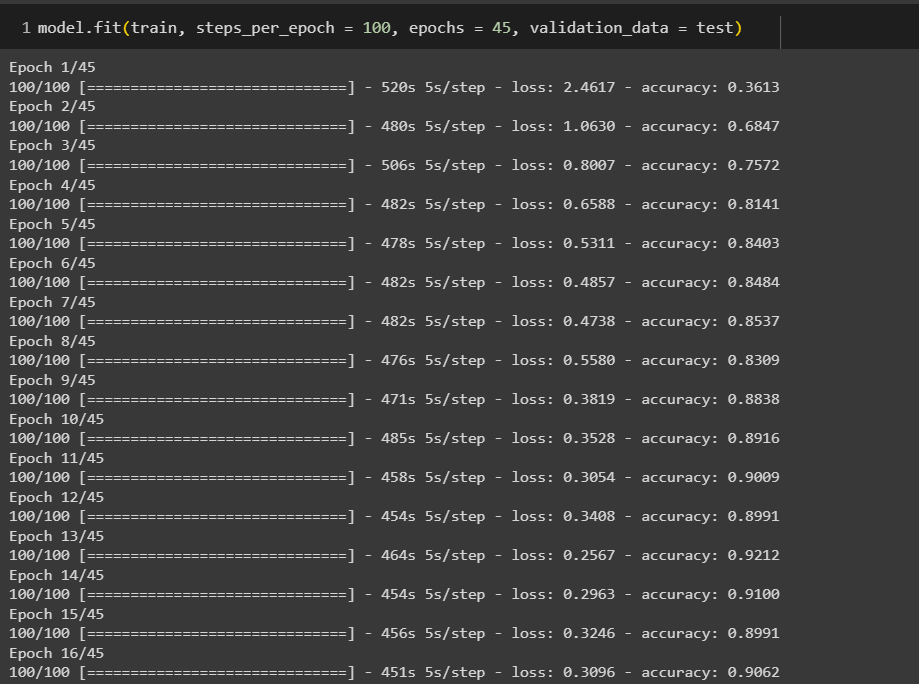
* Summary of the Model

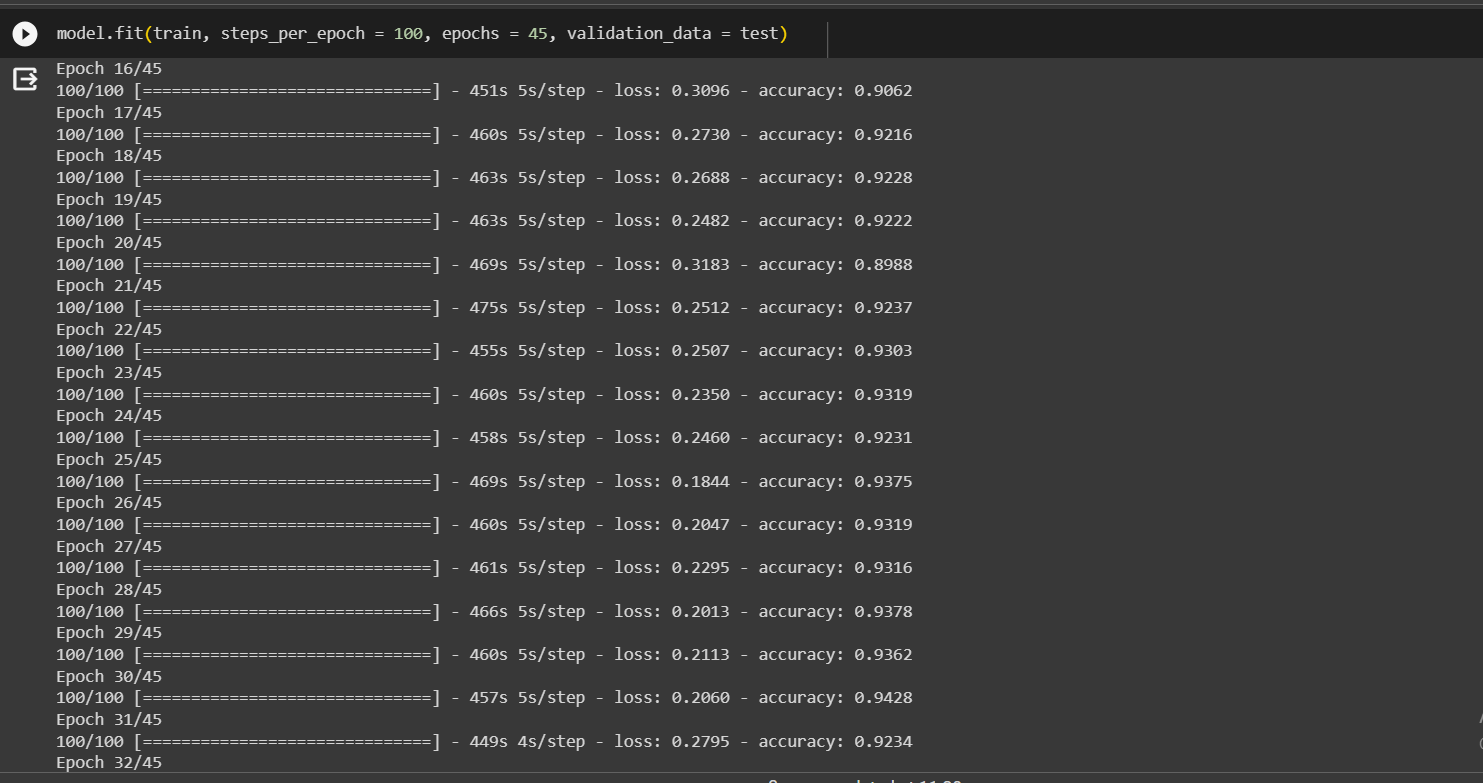
****

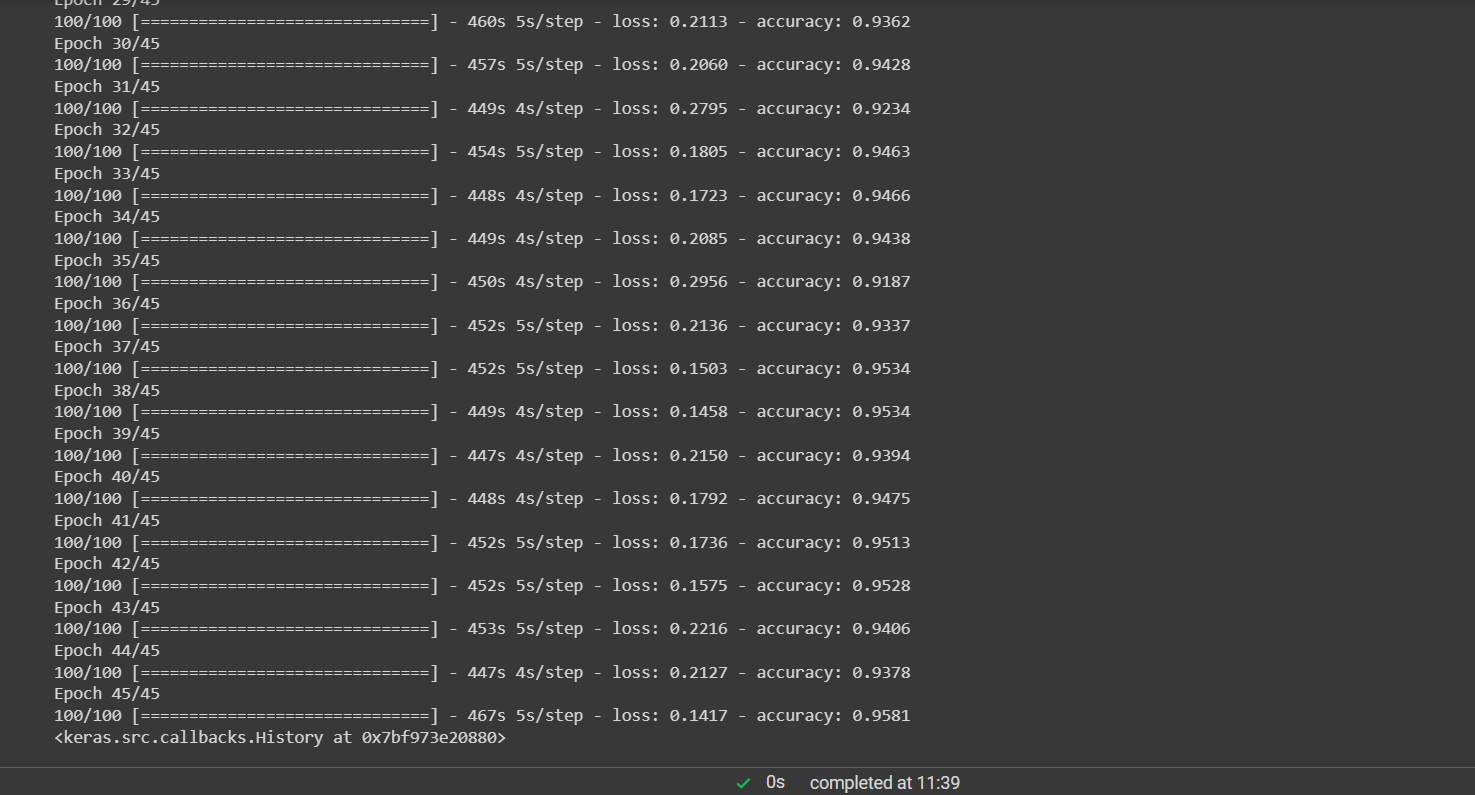
* Compile the model using optimizer and metrics.



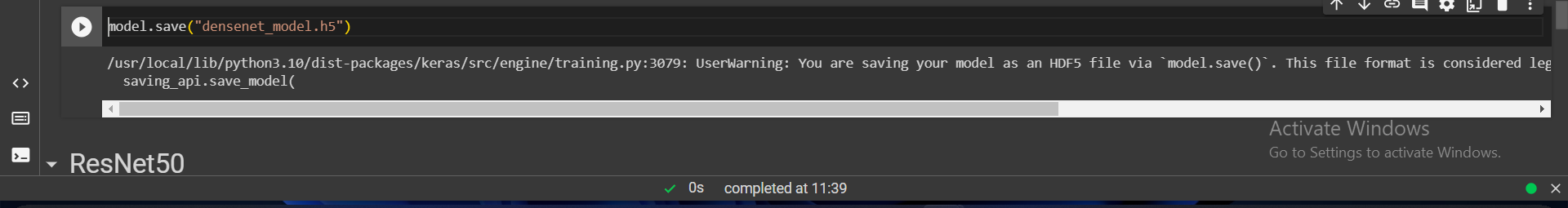
**5. MODEL TRAINING**



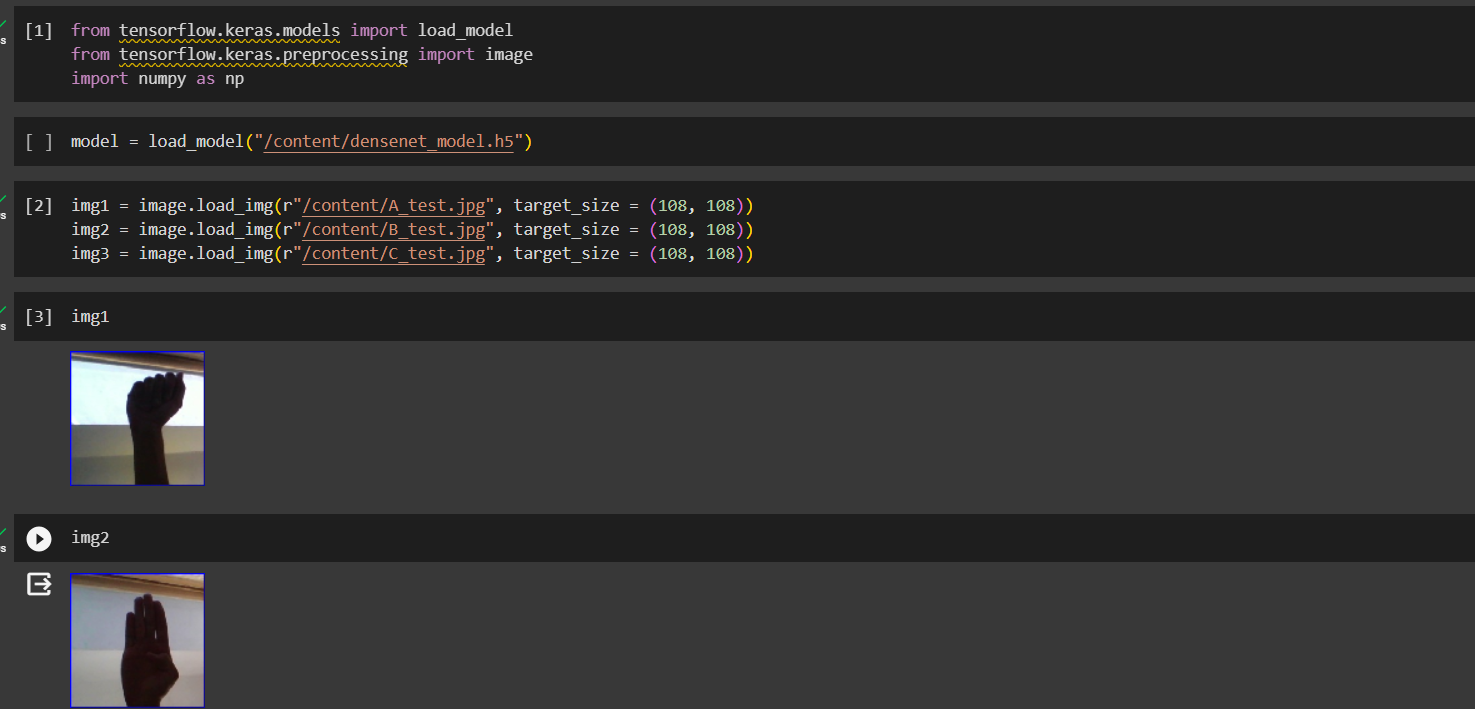


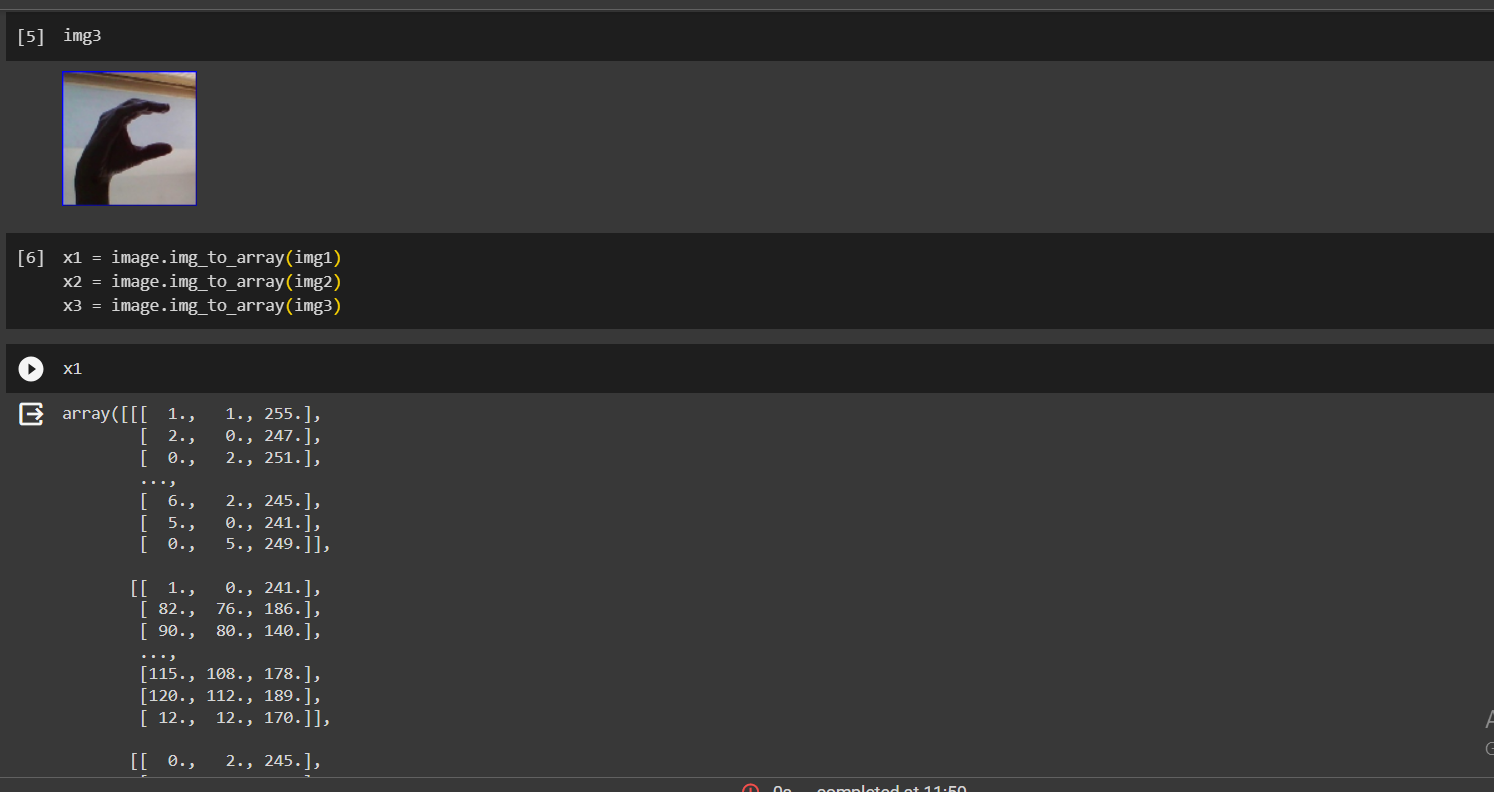


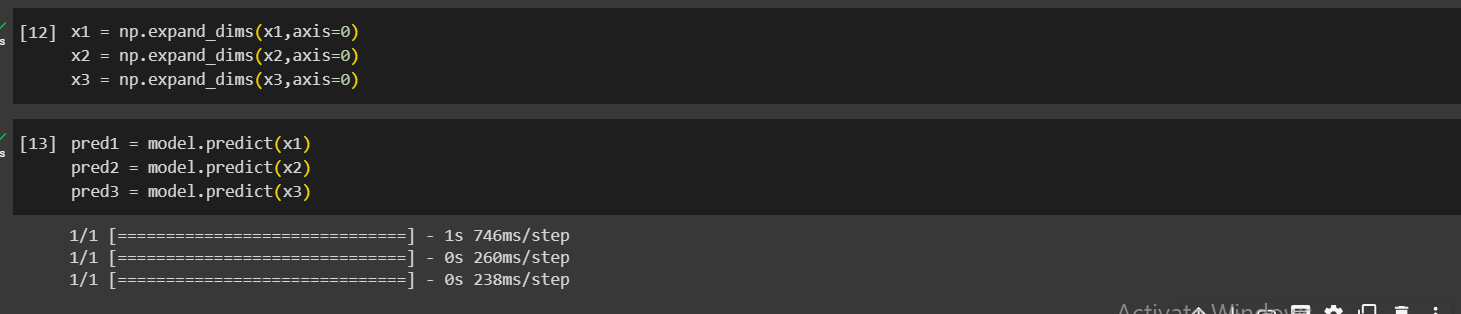
* Save the model

****

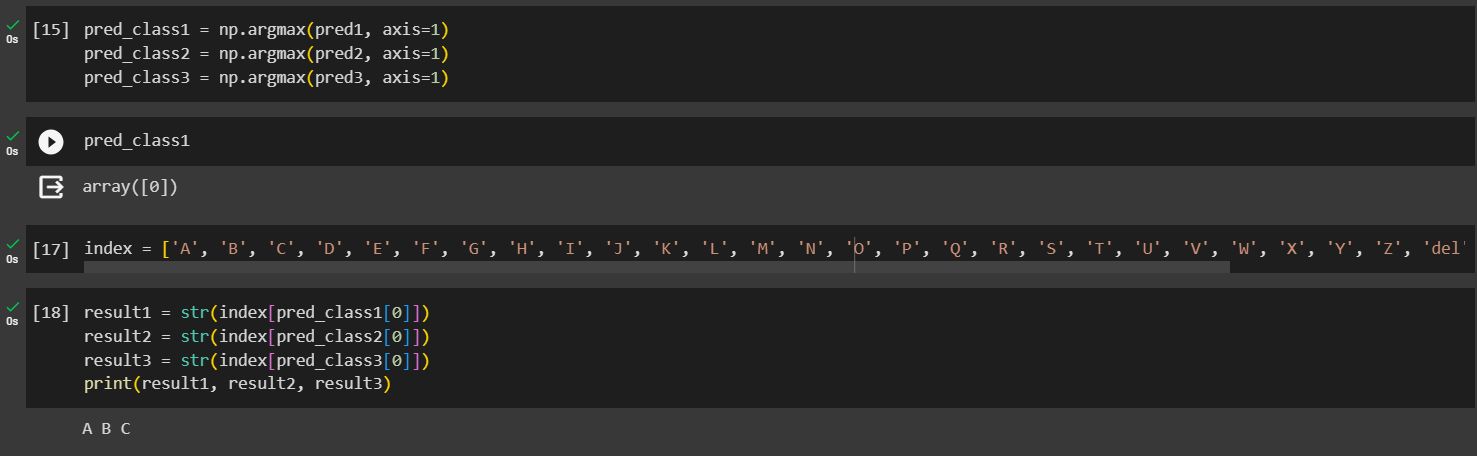
**6. LOAD AND TEST THE MODEL**

****

****

****

**Output Prediction:**

****

**7. APPLICATION BUILDING**

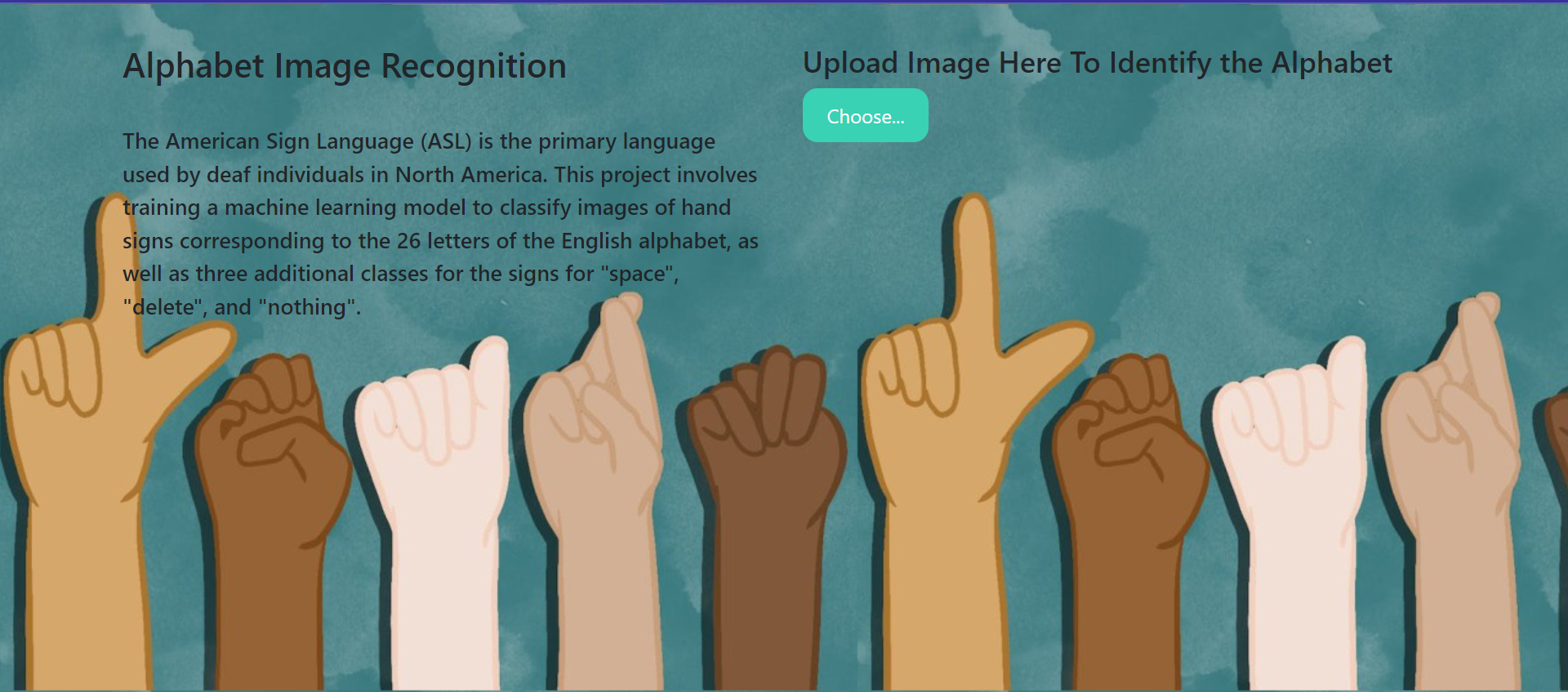
Building our flask application which will be running in our local browser with a user interface. In the flask application, the input(image) is taken from the HTML page. This input is given to the model to predict the alphabet class. Whenever the user interacts with the UI and selects the “Choose…” button, a dialog box of the computer’s files open enabling them to choose an image. Our website page accepts .png, .jpg amd.jpeg file extensions. After choosing the image, a “Predict” button appears, and on clicking the “Predict” button, the letter is predicted

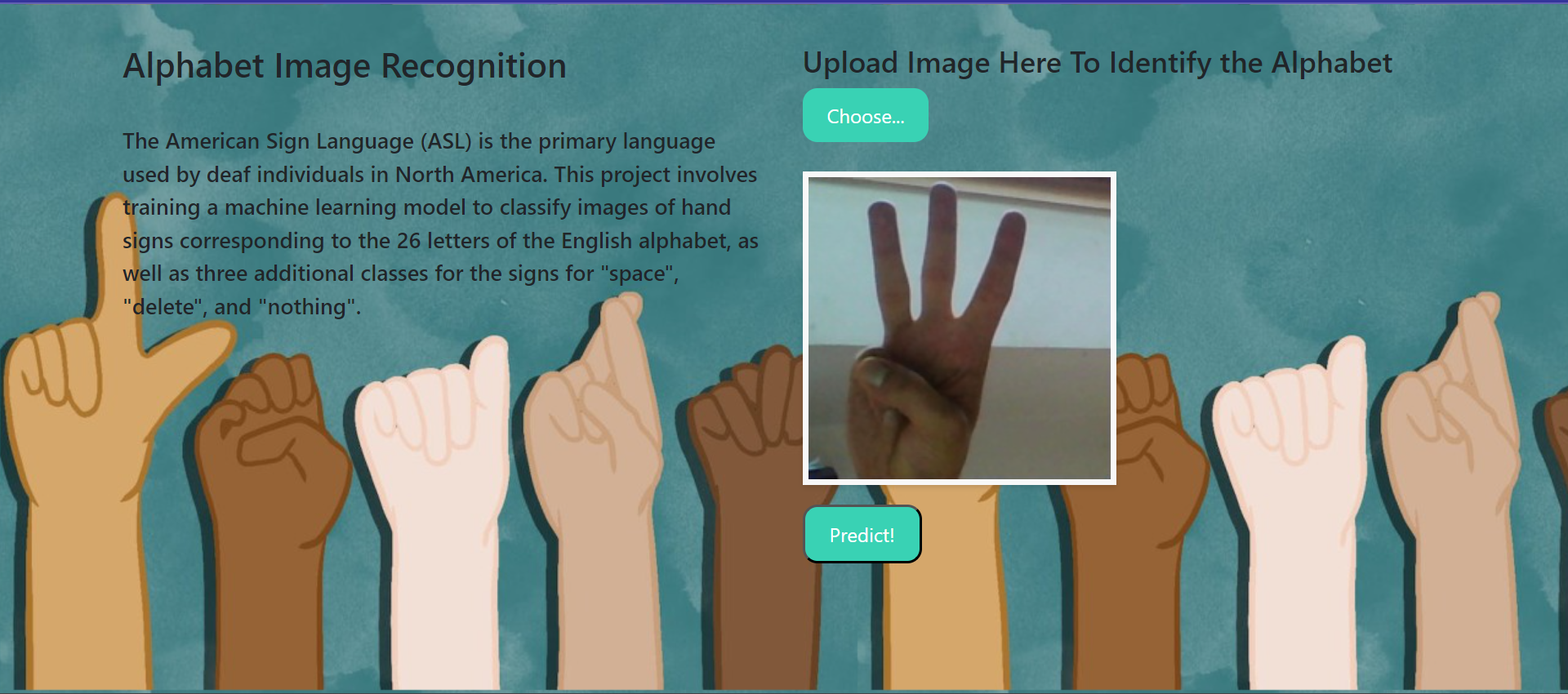
Activity 1: Create HTML Page

o We used HTML - index.html to create the front end part of the web page.

o We also used JavaScript - main.js and CSS - main.css to enhance our functionality and view of HTML pages respectively.

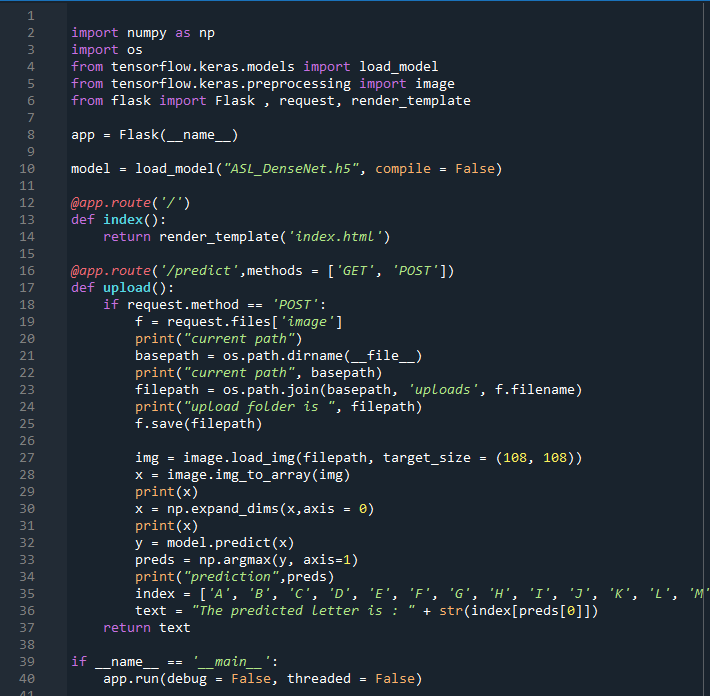
The flow of using our website:







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 two routes (‘/’ - index.html, ‘/predict’ - for prediction on index.html). The main prediction functionality is implemented in the '/predict' route, where the uploaded image is loaded, pre-processed, and passed through the model for prediction. The predicted result is then displayed on the index page. The app can be run by executing the script, and it will start a local server accessible through a web browser.



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.