Dog Breed Identification using Transfer Learning

Team ID : LTVIP2025TMIDS63802

Team Leader : Pammi Likhitha

Team member : Puppalla Rama Krishna

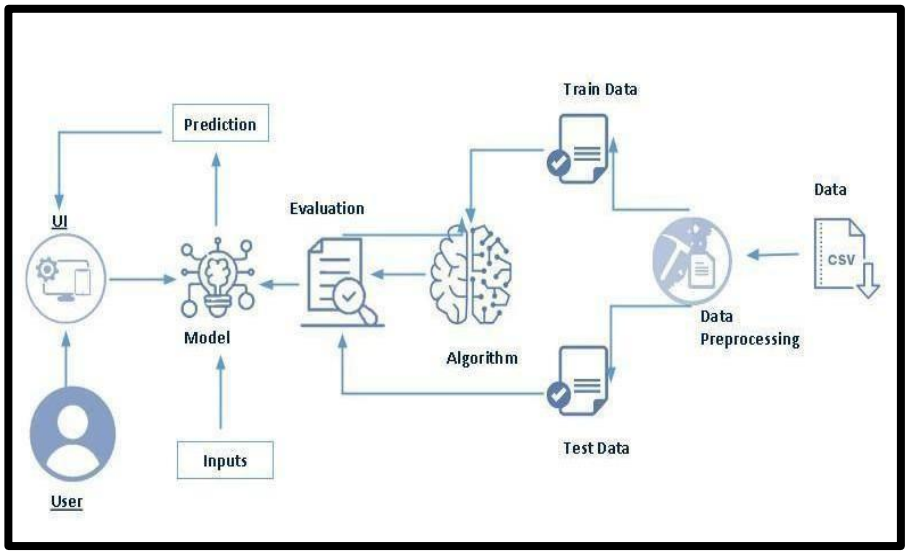
Team member : Shaik Shamila

Team member : Shaik Mahaboob Subhani

Dog Breed Identification using Transfer Learning

Dog Breed Identification using Transfer Learning" aims to develop a robust machine learning model for accurately classifying dog breeds from images. The project leverages transfer learning, a technique that utilizes pre-trained deep learning models as feature extractors, to overcome the challenges of limited training data and computational resources. By fine-tuning a pre-trained convolutional neural network (CNN) on a dataset of dog images, the model learns to distinguish between different breeds with high accuracy. The resulting system provides a valuable tool for dog breed recognition in various applications, including pet care, veterinary medicine, and animal welfare.

Technical Architecture:



Project Objectives

By the end of this project, you will:

- Understand the fundamental concepts and architecture of Convolutional Neural Networks (CNNs) and their role in feature extraction and image classification tasks.
- Gain a broad understanding of image data, including its structure, representation, and challenges such as variability in lighting, pose, and background.
- Learn to preprocess and clean image data using various techniques such as image resizing, normalization, augmentation, and data splitting to improve model performance and generalization.
- Apply transfer learning techniques by leveraging pre-trained CNN models (e.g., Xception, VGG16, or ResNet) to efficiently classify dog breeds with limited training data.
- Fine-tune deep learning models to adapt to specific datasets and optimize accuracy through hyperparameter tuning and performance evaluation.
- Develop and deploy a web application using the Flask framework that allows users to upload images and receive real-time predictions of dog breeds.
- Gain hands-on experience in model integration and deployment, bridging the gap between machine learning models and user-friendly applications.
- Enhance problem-solving and analytical skills by addressing real-world challenges in computer vision using modern AI techniques.

Project Flow

The project follows a structured workflow to build, train, and deploy a deep learning model capable of identifying dog breeds from images through a Flask-based web application.

1. **User Interaction**
    - The user accesses the **Flask web interface** to upload or choose an image of a dog.
    - The uploaded image is sent to the backend server for analysis and prediction.

2. **Model Integration**
    - The backend integrates a **pre-trained CNN model** (e.g., VGG19, Xception, or ResNet) using **Transfer Learning**.
    - The model processes the uploaded image, extracts visual features, and predicts the most probable dog breed.
    - The predicted result is then displayed dynamically on the Flask web UI.

3. **Core Implementation Steps**
   To achieve this, the following stages are completed systematically:

A. Data Collection

- Gather a dataset of labeled dog breed images (e.g., from Kaggle's Dog Breed Identification dataset).
- Organize the dataset into appropriate folders for **training** and **testing**.

B. Data Preparation and Preprocessing

- Create separate **Train** and **Test** directories for model input.
- Import the **ImageDataGenerator** library from Keras for efficient data handling.
- Configure the ImageDataGenerator class with transformations like rotation, flipping, rescaling, and zooming to enhance dataset diversity.
- Apply the configured generators to both training and testing datasets to normalize images and avoid overfitting.

C. Model Building using Transfer Learning

- Import the required **deep learning libraries** (TensorFlow, Keras, NumPy, etc.).
- Load a **pre-trained CNN model** (e.g., **VGG19**) without its top layers to use as a feature extractor.
- Initialize and **add fully connected layers** (Dense, Dropout, Flatten) to customize the model for dog breed classification.
- Configure the learning process using an appropriate **optimizer (Adam/SGD)**, **loss function (categorical cross-entropy)**, and **evaluation metric (accuracy)**.
- Train the model using the prepared dataset while monitoring training and validation performance.
- Evaluate the model on test data to verify generalization and classification accuracy.
- **Save the trained model** (.h5 format) for deployment.

D. Application Development using Flask

- Develop a **Flask-based web application** that connects the front-end and the trained model.
- Design an **HTML template** (e.g., base.html) for user interaction, featuring an image upload option and display area for prediction results.

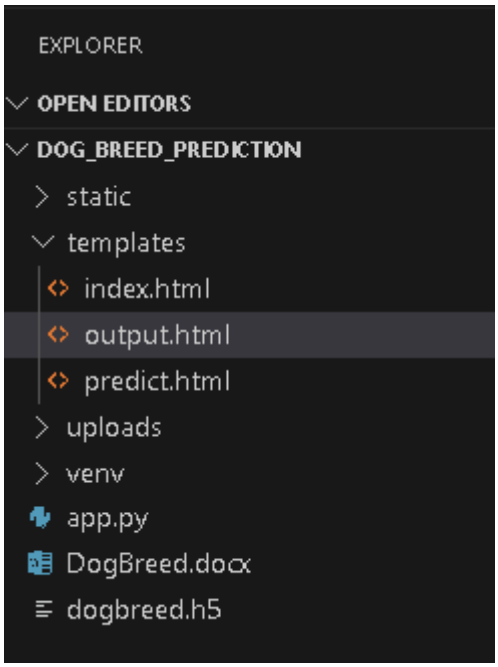
- Write **Python Flask backend code** to handle image uploads, preprocess input images, load the saved model, and generate predictions.
- Integrate model output with the UI to display the predicted dog breed clearly and interactively.

4. **Deployment and Testing**

- Test the complete web application using sample images to ensure accurate predictions.
- Optimize for performance, scalability, and user experience.
- (Optional) Deploy the app on platforms such as **Heroku**, **Render**, or **AWS** for public access.

Project Structure

Create a Project folder which contains files as shown below.



- We are building a Flask Application that needs HTML pages stored in the templates. folder and a python script app.py for server-side scripting
- we need the model which is saved and the saved model in this content is a dogbreed.h5
- templates folder contains index.html, predict.html & output.html pages.
- The Static folder contains css file, images.

Data Collection & Image Preprocessing:

In this milestone First, we will collect images of Dog Breeds then organized into subdirectories based on their respective names as shown in the project structure.

Create folders of types of Dog Breeds that need to be recognized. In this project,

we have collected images of 20 types of Images like affenpinscher, beagle, appenzeller, basset, bluetick, boxer, cairn, doberman, german_shepherd, golden_retriever, kelpie, komondor, leonberg, mexican_hairless, pug, redbone, shih-tzu, toy_poodle, vizsla, whippet they are saved in the respective sub directories with their respective names.

Download the Dataset - https://www.kaggle.com/competitions/dog-breed-identification/data?select=train

In Image Processing, we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

1: Organizing the Images into Different Classes.

```
[ ]  import os
     import shutil
     import sys
```

```
[ ]  dataset_dir = '/content/train'
     labels = pd.read_csv('/content/labels.csv')
```

```
[ ]  import os

     def make_dir(x):
         if os.path.exists(x)==False:
             os.makedirs(x)

     base_dir = './subset'
     make_dir(base_dir)
```

```
[ ]  train_dir = os.path.join(base_dir, 'train')
     make_dir(train_dir)
```

```
[ ]  breeds = labels.breed.unique()
     for breed in breeds:
         # Make folder for each breed
         _ = os.path.join(train_dir, breed)
         make_dir(_)

         # Copy images to the corresponding folders
         images = labels[labels.breed == breed]['id']
         for image in images:
             source = os.path.join(dataset_dir, f'{image}.jpg')
             destination = os.path.join(train_dir, breed,f'{image}.jpg')
             shutil.copyfile(source, destination)
```

- The Images should be organized based on the Image id's.
- So that Training the Model will be simpler.
- For each image ID (image_id) in the current breed, the source path is formed by combining the dataset_dir and the image filename (f'{image_id}.jpg').
- The destination path is formed by combining the breed_folder and the same image filename. shutil.copyfile() is used to copy the image from the source path to the destination path.

Step 2: Import the Image Data Generator Library

In deep learning and computer vision tasks, the amount and diversity of training data play a critical role in determining model performance. However, collecting and labeling large datasets can be time-consuming, expensive, and sometimes impractical.
To overcome this limitation, **Image Data Augmentation** is used — a technique that **artificially increases the size and variability** of a training dataset by applying random transformations to existing images.

This helps the model learn **more generalized patterns** and reduces **overfitting**, allowing it to perform better on unseen data.

```
#import image datagenerator library
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Step 3: Configure the ImageDataGenerator Class

After importing the ImageDataGenerator class from **TensorFlow Keras**, the next crucial step is to **configure it** to specify how image data should be augmented during training. The configuration defines the **types and intensity** of transformations that will be applied to the training images to improve model robustness and generalization.

In deep learning projects involving images, especially when using a **limited dataset**, data augmentation plays a vital role in:

- Increasing dataset diversity
- Reducing overfitting
- Improving the model's ability to recognize patterns under varied conditions

```
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
```

An instance of the Image Data Generator class can be constructed for train and test.

4: Apply ImageDataGenerator functionality to Trainset and Test set:

Let us apply ImageDataGenerator functionality to Train set and Test set by using the following code. For Training set using flow_from_directory function.

This function will return batches of images from the subdirectories affenpinscher, beagle, appenzeller, basset, bluetick, boxer, cairn, doberman, german_shepherd, golden_retriever, kelpie, komondor, leonberg, mexican_hairless, pug, redbone, shih-tzu, toy_poodle, vizsla, whippe, together with labels 0 to 19.

Arguments:

- directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
- batch_size: Size of the batches of data which is 32.
- target_size: Size to resize images after they are read from disk.
- class_mode:

‘int': means that the labels are encoded as integers (e.g. for sparse_categorical_crossentropy loss).

'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical_crossentropy loss).

- 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary_crossentropy).
- None (no labels).

```
[ ]  datagen = ImageDataGenerator()
     generator = datagen.flow_from_directory(
         dataset,
         target_size=(224, 224),  # Adjust target size as needed
         batch_size=32,
         class_mode='categorical',
         shuffle=False,  # Ensure order is maintained for class indices
         classes=selected_classes  # Specify the selected classes
     )

     Found 1683 images belonging to 20 classes.
```

```
test_datagen = ImageDataGenerator(rescale=1./255)
```

Model Building

Now it's time to build our Convolutional Neural Networking using vgg19 which contains an input layer along with the convolution, max-pooling, and finally an output layer.

Link: https://thesmartbridge.com/documents/spsaimldocs/CNNflow.pdf

1: Importing the Model Building Libraries

Before building and training a Convolutional Neural Network (CNN) for **Dog Breed Identification**, it is essential to import the necessary libraries that provide functions and classes for deep learning, image processing, and model evaluation.

The libraries form the backbone of the entire model development pipeline — from loading and preprocessing data to building, training, and saving the final model.

Key Libraries Used:

1. **TensorFlow and Keras**

- TensorFlow is a powerful open-source framework widely used for developing and deploying deep learning models.
- Keras, integrated within TensorFlow, provides a **high-level API** that simplifies neural network creation, allowing quick experimentation with model architectures.
- These libraries provide tools for building CNNs, applying transfer learning, and training models efficiently.
-

Importing Libraries

```
[23] import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras.preprocessing.image  import ImageDataGenerator
     from tensorflow.keras.layers import Dense
     from tensorflow.keras.activations import softmax
     from keras.api._v2.keras import activations
```

2: Importing the VGG19 Model

The **VGG19** model is one of the most popular and powerful **Convolutional Neural Network (CNN)** architectures developed by the **Visual Geometry Group (VGG)** at the University of Oxford. It consists of **19 layers** (16 convolutional layers and 3 fully connected layers) and is known for its simple, uniform architecture using **3×3 convolution filters** stacked on top of each other, which enables effective feature extraction from images.

In this project, **VGG19** is used as the **base model** for transfer learning to perform **Dog Breed Identification**. Instead of training a CNN from scratch — which requires a large dataset and high computational resources — we leverage **pre-trained weights from the ImageNet dataset**, which contains over **1.2 million images** across **1,000 object categories**.

These pre-trained weights help the model recognize low-level features like edges, textures, and shapes, which are transferable and useful for identifying dog breeds.

```
from tensorflow.keras.applications.vgg19 import VGG19
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

3: Initializing the model:

- The model will be initialized with the pre-trained weights from the ImageNet dataset, and the last fully connected layer will be excluded from the model architecture.
- The loop that follows freezes the weights of all the layers in the VGG19 model by setting

`i.trainable=False` for each layer in the model. This is done to prevent the weights from being updated during training, as the model is already pre-trained on a large dataset.

- Finally, a `Flatten()` layer is added to the output of the VGG19 model to convert the output tensor into a 1D tensor.

- The resulting model can be used as a feature extractor for transfer learning or as a starting point for building a new model on top of it.

```
[ ] Image_size=[224,224]
```

```
[ ] sol=VGG19(input_shape=Image_size + [3] , weights='imagenet' , include_top = False)
```

```
[ ] for i in sol.layers:
      i.trainable = False
```

```
[ ] y=Flatten()(sol.output)
```

4: Adding Fully connected Layers

- For information regarding CNN Layers refer to the link
  Link: https://victorzhou.com/blog/intro-to-cnns-part-1/
- As the input image contains three channels, we are specifying the input shape as (128,128,3).

- We are adding a output layer with activation function as “softmax”.

```
[ ] from keras.api._v2.keras import activations
    final = Dense(20, activation = 'softmax')(y)
```

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer. The number of neurons in the Dense layer is the same as the number of classes in the training set.The neurons in the last De nse layer, use softmax activation to convert their outputs into respective probabilities.

Understanding the model is a very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.

```
vgg19_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv4 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808
```

5: Configure The Learning Process

- The compilation is the final step in creating a model. Once the compilation is done, we

can move on to the training phase. The loss function is used to find errors or deviations inthe learning process. Keras requires a loss function during the model compilation process.

- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer.

- Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process.

```
vgg19_model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['Accuracy'])
```

6: Train The model

Now, let us train our model with our image dataset. The model is trained for 6 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 30 epochs and probably there is further scope to improve the model.

fit_generator functions used to train a deep learning neural network.
Arguments:

- steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.

- Epochs: an integer and number of epochs we want to train our model for.

- validation_data can be either:
  - an inputs and targets list
  - a generator
  - an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.

- validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is

stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

```
[ ]  vgg19_model.fit(generator, epochs = 6)
```

```
Epoch 1/6
53/53 [==============================] - 1353s 25s/step - loss: 126.8239 - Accuracy: 0.1218
Epoch 2/6
53/53 [==============================] - 1352s 26s/step - loss: 30.5051 - Accuracy: 0.6049
Epoch 3/6
53/53 [==============================] - 1336s 25s/step - loss: 3.8852 - Accuracy: 0.9008
Epoch 4/6
53/53 [==============================] - 1333s 25s/step - loss: 1.1688 - Accuracy: 0.9584
Epoch 5/6
53/53 [==============================] - 1332s 25s/step - loss: 1.6495 - Accuracy: 0.9548
Epoch 6/6
53/53 [==============================] - 1287s 24s/step - loss: 0.2561 - Accuracy: 0.9857
<keras.callbacks.History at 0x7f66172dd540>
```

7: Save the Model

The model is saved with .h5 extension as follows.

An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

```
[ ]  vgg19_model.save("dogbreed.h5")
```

8: Test The model

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.

```
[ ] import numpy as np
    from keras.preprocessing import image
    from keras.applications.vgg16 import preprocess_input
    from tensorflow.keras.preprocessing.image import load_img, img_to_array

[ ] img_path ='/content/test/000621fb3cbb32d8935728e48679680e.jpg'

[ ] img = load_img(img_path, target_size=(224, 224))
    x = img_to_array(img)
    x = preprocess_input(x)
    preds = vgg19_model.predict(np.array([x]))

    1/1 [==============================] - 1s 1s/step

[ ] class_names=['affenpinscher','beagle','appenzeller','basset','bluetick','boxer','cairn','doberman','german_shepherd','golden_retriever','kelpie','komondor','leonberg',

[ ] class_names[np.argmax(preds)]
```

Taking an image as input and checking the results

By using the model we are predicting the output for the given input image. The predicted class index name will be printed here.

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 Colon Diseases and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the “Inspect” button, the next page is opened where the user chooses the image and predicts the output.

1 : Create HTML Pages

- We use HTML to create the front end part of the web page.
- Here, we have created 3 HTML pages- index.html, predict.html, and output.html
- home.html displays the home page.
- index.html displays an introduction about the project
- upload.html gives the emergency alert For more information regarding HTML https://www.w3schools.com/html/
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.
- Link :CSS , JS

2: Building Python Code

After designing the front-end (HTML templates) and saving the trained model (dogbreed.h5), the next step is to **build the Python backend code** using the **Flask framework**.
This Python code acts as the **server-side logic** that connects the **user interface**, **deep learning model**, and **image prediction pipeline**.

The app.py script serves as the core of the application — handling requests, processing uploaded images, loading the trained model, and displaying predictions through the web interface.

1: Importing Libraries

The first step is usually importing the libraries that will be needed in the program.

Importing the flask module in the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (name) as argument Pickle library to load the model file.

```
import numpy as np
import os
import tensorflow as tf
from PIL import Image
from flask import Flask, render_template,request,jsonify,url_for,redirect
from tensorflow.keras.preprocessing.image import load_img,img_to_array
```

2: Creating the Flask Application and Loading the Model

After building and training the CNN model for dog breed classification, the next crucial step is to **integrate the model into a Flask web application**. This allows end-users to interact with the model in real time, uploading images through a web interface and receiving predictions immediately.

This step involves two key tasks:

1. **Creating the Flask Application**
2. **Loading the Pre-Trained Model Using load_model**

```
app=Flask(__name__)
model = tf.keras.models.load_model('dogbreed.h5')
```

3: Routing to the html Page

Here, the declared constructor is used to route to the HTML page created earlier. In the above example, '/' URL is bound with index.html function. Hence, when the home page of a web server is opened in the browser, the html page will be rendered. Whenever you browse an image from the html page this photo can be accessed through POST or GET Method.

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

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

Showcasing prediction on UI:

```
@app.route('/output',methods=['GET','POST'])
def output():
    if request.method=='POST':
        f=request.files['file']
        basepath=os.path.dirname(__file__)
        filepath=os.path.join(basepath,'uploads',f.filename)
        f.save(filepath)
        img=load_img(filepath,target_size=(224,224))
        # Resize the image to the required size

        # Convert the image to an array and normalize it
        image_array = np.array(img)
        # Add a batch dimension
        image_array = np.expand_dims(image_array, axis=0)
        # Use the pre-trained model to make a prediction
        pred=np.argmax(model.predict(image_array),axis=1)
        index=['affenpinscher','beagle','appenzeller','basset','bluetick','boxer','cairn','doberman','german_shepherd','golden_retriever','
        prediction = index[int(pred)]
        print("prediction")
        return render_template("output.html",predict = prediction)
```

Here we are defining a function which requests the browsed file from the html page using the post method. The requested picture file is then saved to the uploads folder in this same directory using OS library. Using the load image class from Keras library we are retrieving the saved picture from the path declared. We are applying some image processing techniques and then sending that preprocessed image to the model for predicting the class. This returns the numerical value of a class (like 0 to 19.) which lies in the 0th index of the variable preds. This numerical value is passed to the index variable declared. This returns the name of the class. This name is rendered to the prediction variable used in the html

page.
Finally, Run the application

This is used to run the application in a local host.

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

3:Run the application

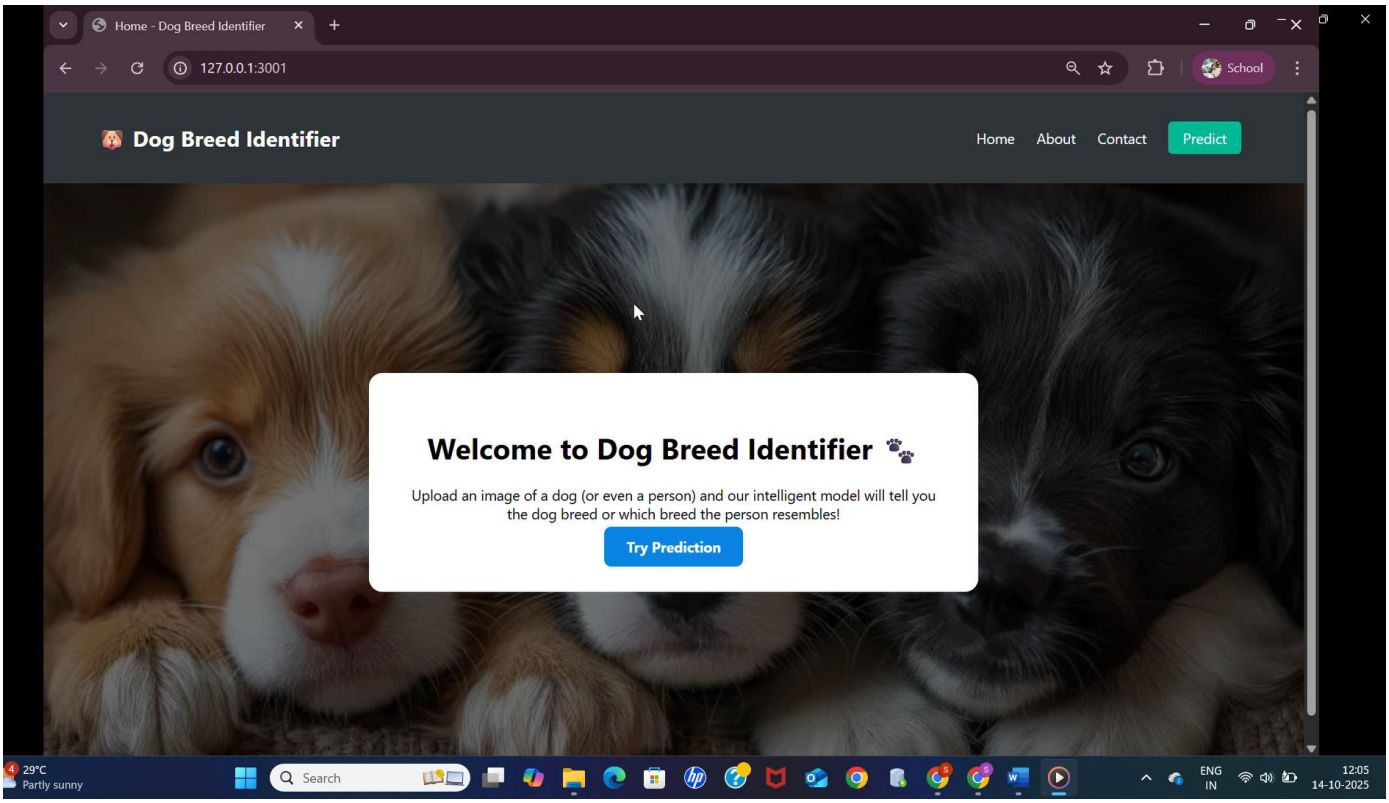
- Open the anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type “python app.py” command.
- It will show the local host where your app is running on http://127.0.0.1.5000/
- Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page.

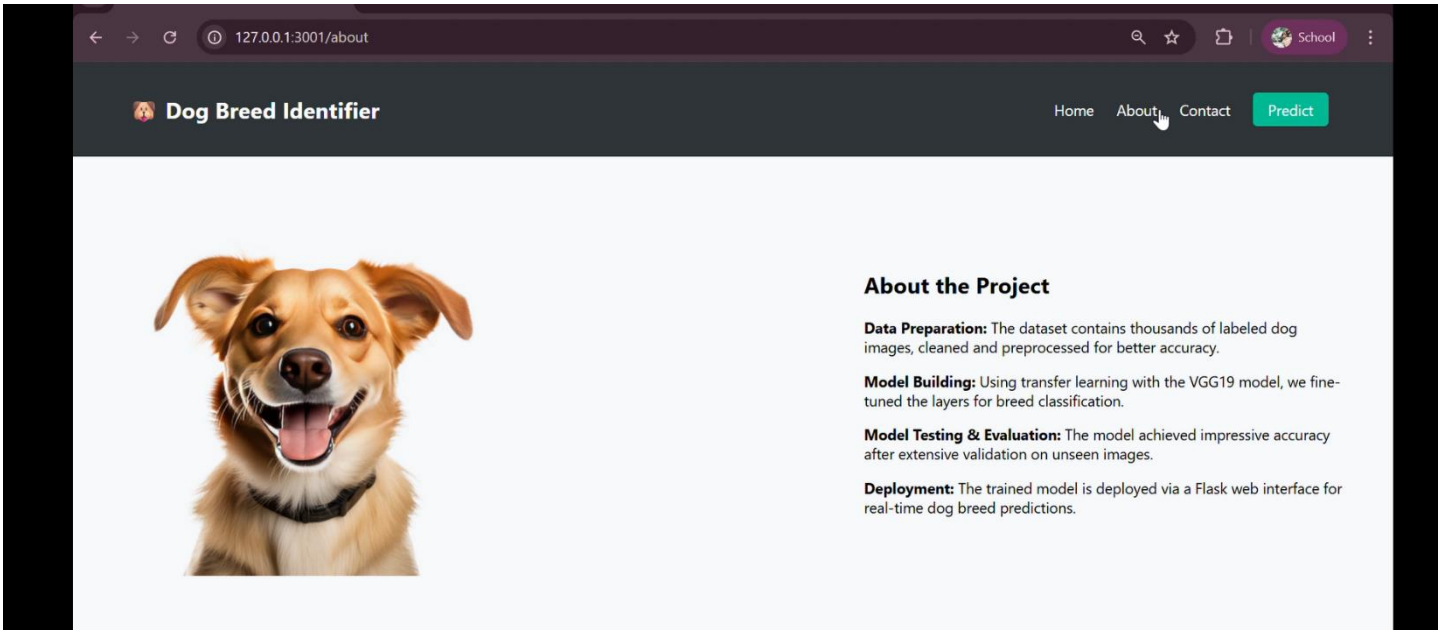
Then it will run on localhost: 5000

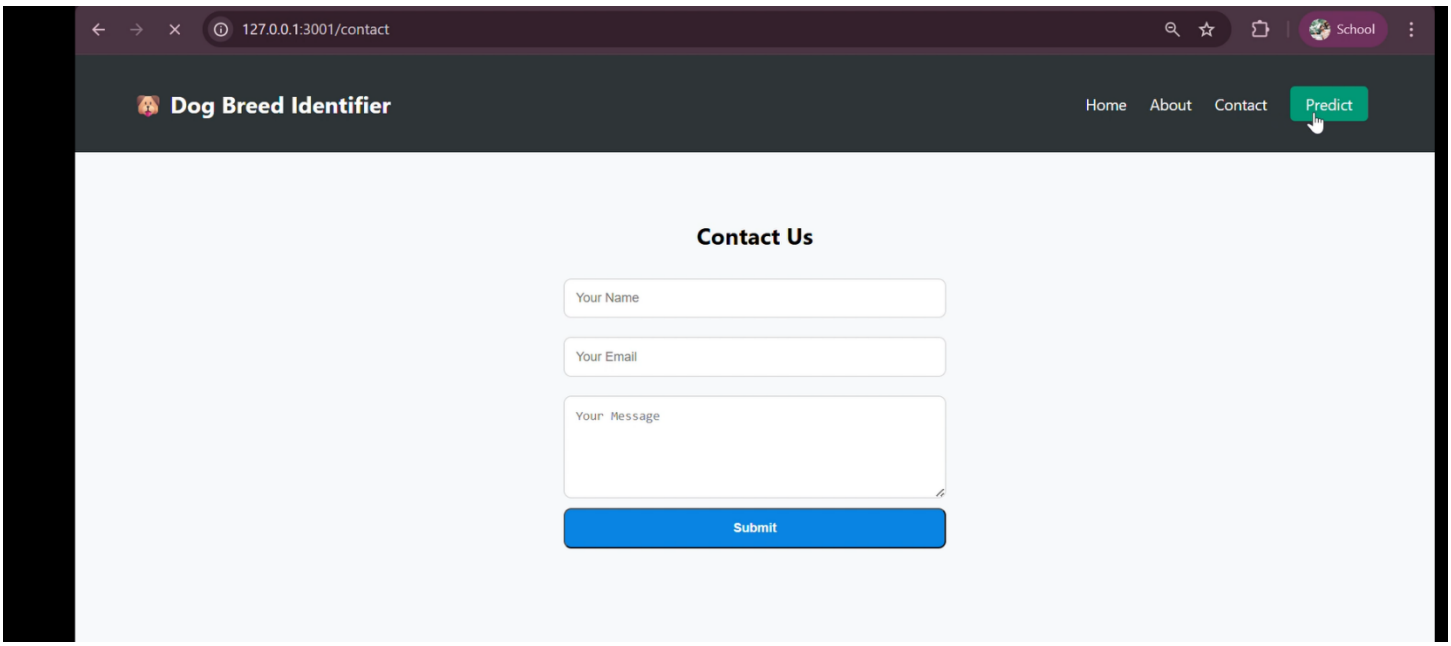
```
PS F:\Smart_Internz\Dog_Breed_prediction> python -u "f:\Smart_Internz\Dog_Breed_prediction\app.py"
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 580-415-876
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [11/Jul/2023 12:15:29] "GET / HTTP/1.1" 200 -
```

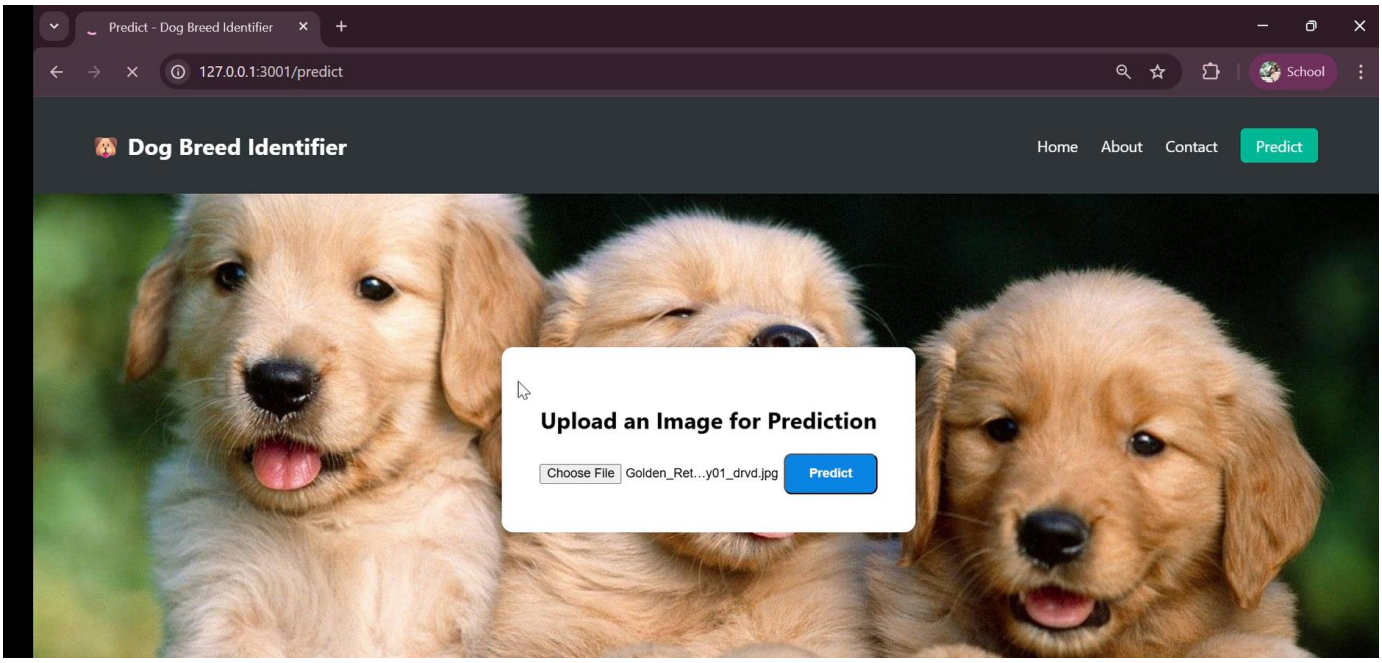
Navigate to the localhost (http://127.0.0.1:5000/)where you can view your web page.

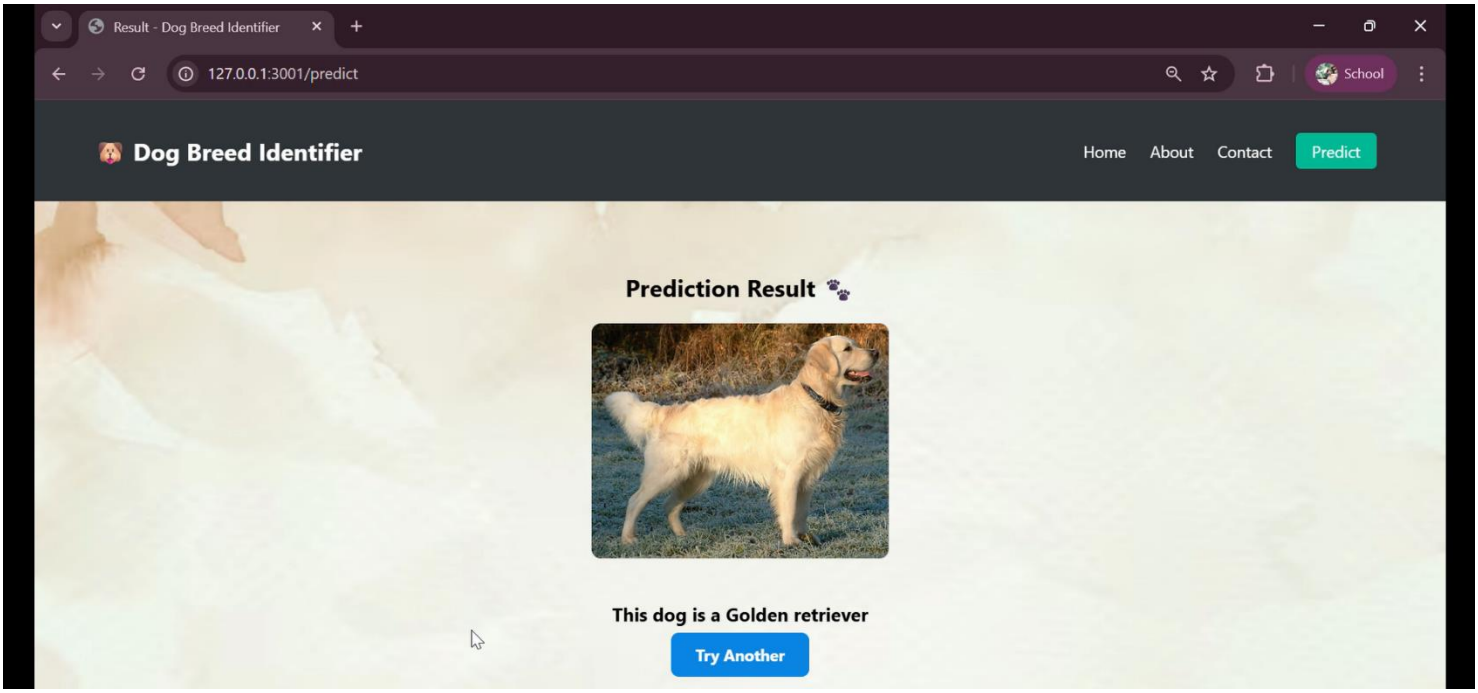
FINAL OUTPUT:

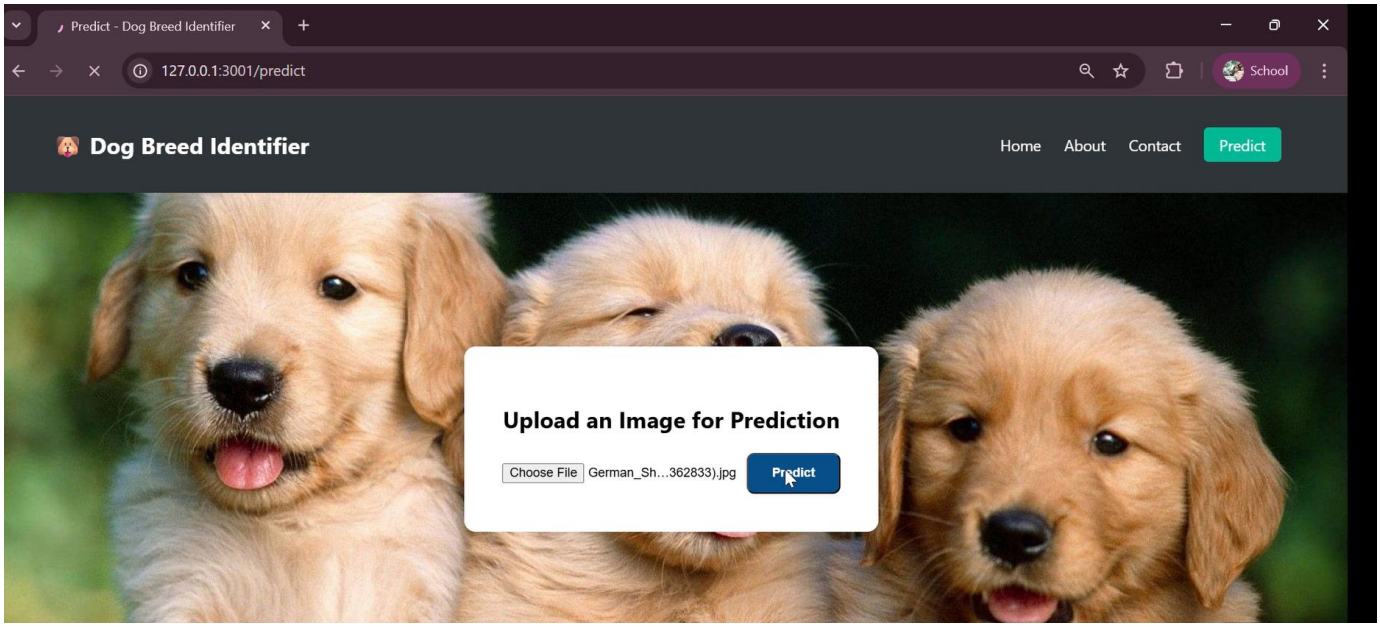


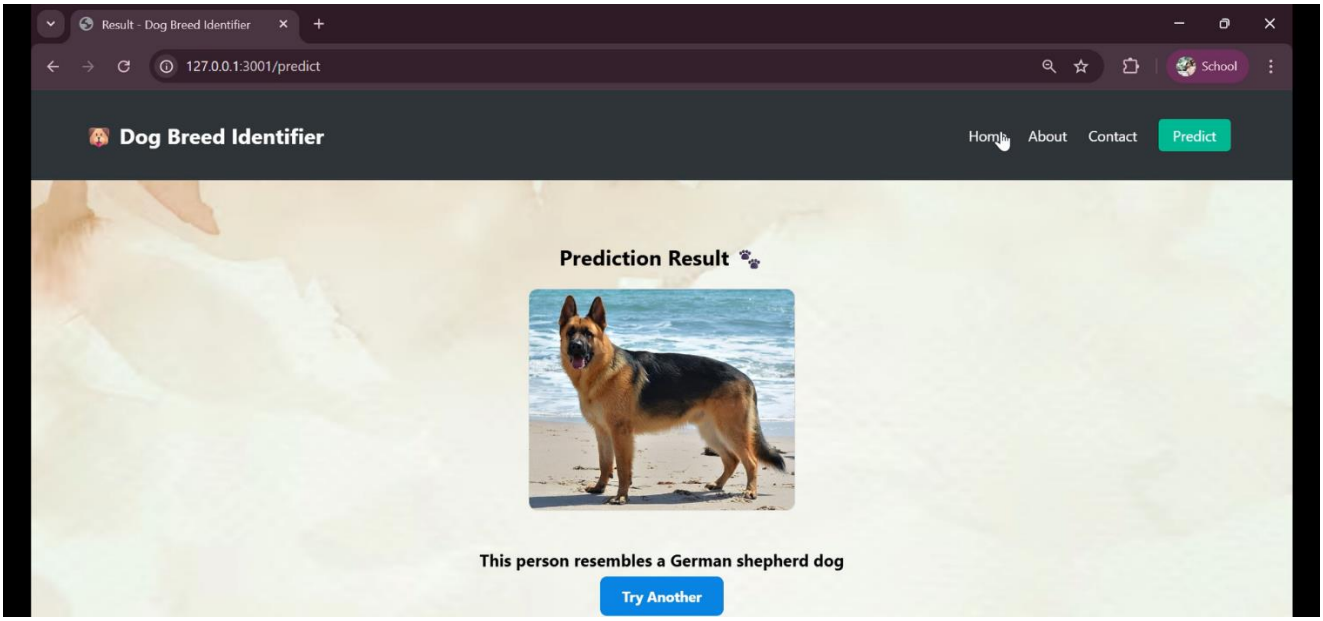



127.0.0.1:3001/contact
School
Dog Breed Identifier
Home
About
Contact
Predict
Contact Us
Your Name
Your Email
Your Message
Submit


Predict - Dog Breed Identifier
127.0.0.1:3001/predict
School
Dog Breed Identifier
Home
About
Contact
Predict
Upload an Image for Prediction
Choose File
Golden_Ret...y01_drvd.jpg
Predict


Result - Dog Breed Identifier
127.0.0.1:3001/predict
School
Dog Breed Identifier
Home
About
Contact
Predict
Prediction Result
This dog is a Golden retriever
Try Another


Predict - Dog Breed Identifier
127.0.0.1:3001/predict
School
Dog Breed Identifier
Home
About
Contact
Predict
Upload an Image for Prediction
Choose File
German_Sh...362833).jpg
Predict



References

- Chollet, F. (2018). *Deep Learning with Python* (2nd Edition). Manning Publications.
  - Practical guide for building CNNs and using transfer learning with Keras and TensorFlow.
- Simonyan, K., & Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv:1409.1556.
  - Original research paper describing the VGG19 architecture used for feature extraction.
- TensorFlow Documentation: Transfer Learning & ImageDataGenerator.
  - https://www.tensorflow.org/tutorials/images/transfer_learning
- Flask Official Documentation.
  - https://flask.palletsprojects.com/en/2.3.x/

GitHUB Link:

https://github.com/likhi2810/Dog-Breed-Identification-using-Transfer-Learning