Dog Breed Identification Using Transfer Learning

Project Description:

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.

Scenarios:

Pet Adoption Platform:

Scenario: An online pet adoption platform wants to improve the user experience by automatically categorizing the dog breeds of animals available for adoption based on uploaded images. This helps potential adopters quickly find dogs that match their preferences.

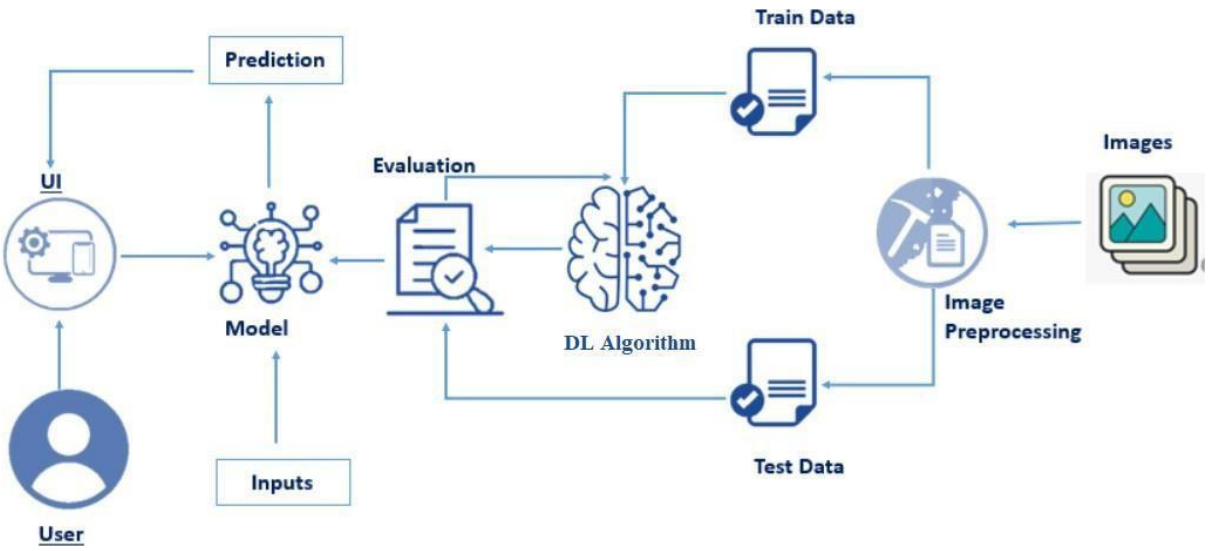
Lost Dog Identification:

Scenario: A person finds a lost dog and wants to help reunite it with its owner. However, they are unsure of the dog's breed, making it challenging to create an accurate description for missing pet posters or online announcements.

Veterinary Diagnosis Support:

Scenario: A veterinarian needs assistance in identifying the breed of a mixed-breed dog brought in for a health checkup. Knowing the breed can provide valuable insights into potential genetic predispositions to certain health conditions or behavioral traits.

Technical Architecture:



Pre requisites:

To complete this project, following software's, concepts and packages are used

- **Anaconda navigator and pycharm:**
  - Refer the link below to download anaconda navigator
  - Link : https://youtu.be/1ra4zH2G4o0
- **Python packages:**
  - Open anaconda prompt as administrator
  - Type "pip install numpy" and click enter.
  - Type "pip install pandas" and click enter.
  - Type "pip install scikit-learn" and click enter.
  - Type "pip install matplotlib" and click enter.
  - Type "pip install scipy" and click enter.
  - Type "pip install pickle-mixin" and click enter.
  - Type "pip install seaborn" and click enter.
  - Type "pip install Flask" and click enter.

Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- **Deep Learning concepts:**
  - CNN: a convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery.

- VGG19: VGG19 is a deep convolutional neural network architecture for image classification, consisting of 19 layers with small convolution filters.
- Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

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 is 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 must complete all the activities and tasks listed below.

- Data Collection.
  - Create Train and Test Folders.

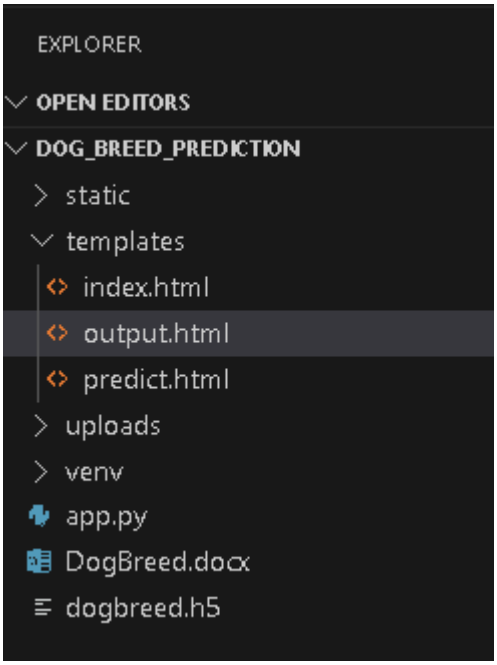
  Data Preprocessing.
  - Import the ImageDataGenerator library
  - Configure ImageDataGenerator class
  - Apply Image Data Generator functionality to Train dataset and Test dataset

  Model Building
  - Import the model building Libraries.
  - Importing the VGG19.
  - Initializing the model
  - Adding Fully connected Layer
  - Configure the Learning Process
  - Training and Testing the model.
  - Save the Model

  Application Building
  - Create an HTML file
  - Build Python Code

Project Structure:

Create the 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.

Milestone 1: Data Collection

ML depends heavily on data, It is most crucial aspect that makes algorithm training possible.

So this section allows you to download the required dataset.

Activity 1: Download the dataset

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.

Milestone 2: Data Preprocessing

Activity 1: Organizing the Images into Different Classes.

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.

The supported breeds are totally 25 they are great_pyrenees, irish_water_spaniel, komondor, labrador_retriever, lhasa, old_english_sheepdog, pug, standard_poodle, afghan_hound, bedlington_terrier, black-and-tan_coonhound, bouvier_des_flandres, bull_mastiff, cardigan, chow, cocker_spaniel, collie, dingo, doberman, english_foxhound, entlebucher, german_shepherd, golden_retriever, gordon_setter, great_dane, great_pyrenees, irish_water_spaniel

```
def build_and_train(dataset_dir='dataset', img_size=(128, 128), batch_size=32, epochs=6, output_model='dogbreed.h5'):
    train_dir = os.path.join(dataset_dir, 'train')
    if not os.path.isdir(train_dir):
        raise FileNotFoundError(f"Training directory not found: {train_dir}")
```

Activity 2: Import the ImageDataGenerator library.

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.
Let us import the ImageDataGenerator class from tensorflow Keras.

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

Activity 3: Configure ImageDataGenerator class

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation. There are five main types of data augmentation techniques for image data; specifically:

- Image shifts via the width_shift_range and height_shift_range arguments.
- The image flips via the horizontal_flip and vertical_flip arguments.
- Image rotations via the rotation_range argument
- Image brightness via the brightness_range argument.
- Image zoom via the zoom_range argument.

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

An instance of the ImageDataGenerator class can be constructed for train and test.

Activity 4: Apply ImageDataGenerator functionality to Trainset and Test set:

```
def build_and_train(dataset_dir='dataset', img_size=(128, 128), batch_size=32, epochs=6, output_model='dogbreed.h5'):
    train_dir = os.path.join(dataset_dir, 'train')
    if not os.path.isdir(train_dir):
        raise FileNotFoundError(f"Training directory not found: {train_dir}")

    datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        validation_split=0.2
    )
```


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:

1. 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.
2. batch_size: Size of the batches of data which is 32.

3. target_size: Size to resize images after they are read from disk.
4. 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).

5. '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).
6. None (no labels).

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

Milestone 3: Model Building

Activity 1: Importing the Model Building Libraries

Importing the necessary libraries

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
```

Activity 2: Importing the VGG19 model:

To initialize the VGG19 model, the weights are usually pre-trained on the ImageNet dataset, which is a large-scale dataset of images belonging to 1,000 different categories. These pre-trained weights can be downloaded from the internet, and they can be used as a starting point to fine-tune the model for a specific task, such as object recognition or classification.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG19
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Flatten, Dense, Dropout, Input
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau
```

We will fill the missing values in numeric data type using mean value of that particular column and categorical data type using the most repeated value.

Activity 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.

```
num_classes = train_generator.num_classes

base_model = VGG19(weights='imagenet', include_top=False, input_shape=(*img_size, 3))
for layer in base_model.layers:
    layer.trainable = False
```

Activity 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”.

```
base_model = VGG19(weights='imagenet', include_top=False, input_shape=(*img_size, 3))
for layer in base_model.layers:
    layer.trainable = False

x = base_model.output
x = Flatten()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
outputs = Dense(num_classes, activation='softmax')(x)
```

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.

```
model = Model(inputs=base_model.input, outputs=outputs)
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
```

Activity 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.

```
model = Model(inputs=base_model.input, outputs=outputs)
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
```

Activity 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.

```
model.fit(
    train_generator,
    epochs=epochs,
    validation_data=val_generator,
    callbacks=callbacks
)

# Save class names (index -> class)
class_indices = train_generator.class_indices
inv_map = {v: k for k, v in class_indices.items()}
with open('class_names.json', 'w') as f:
    json.dump(inv_map, f)

print(f"Training finished. Best model saved to {output_model} and classes to class_names.json")
```

Activity 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")
```

Activity 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.

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.

Milestone 5: 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.
- Building HTML Pages
- Building serverside script

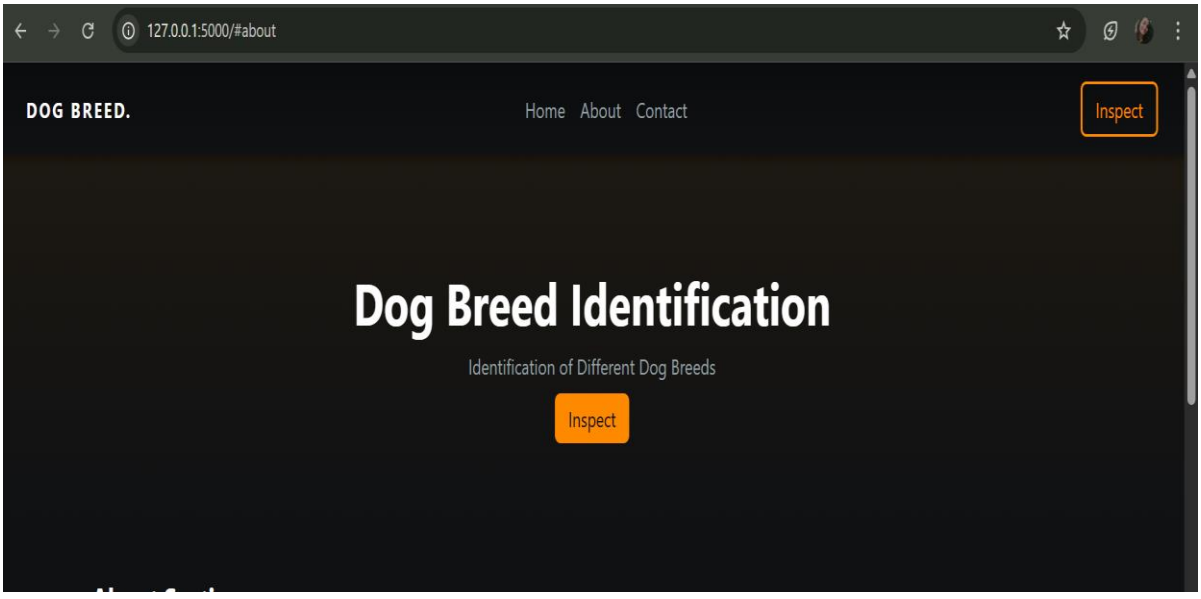
Activity1: Building Html Pages:

- 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

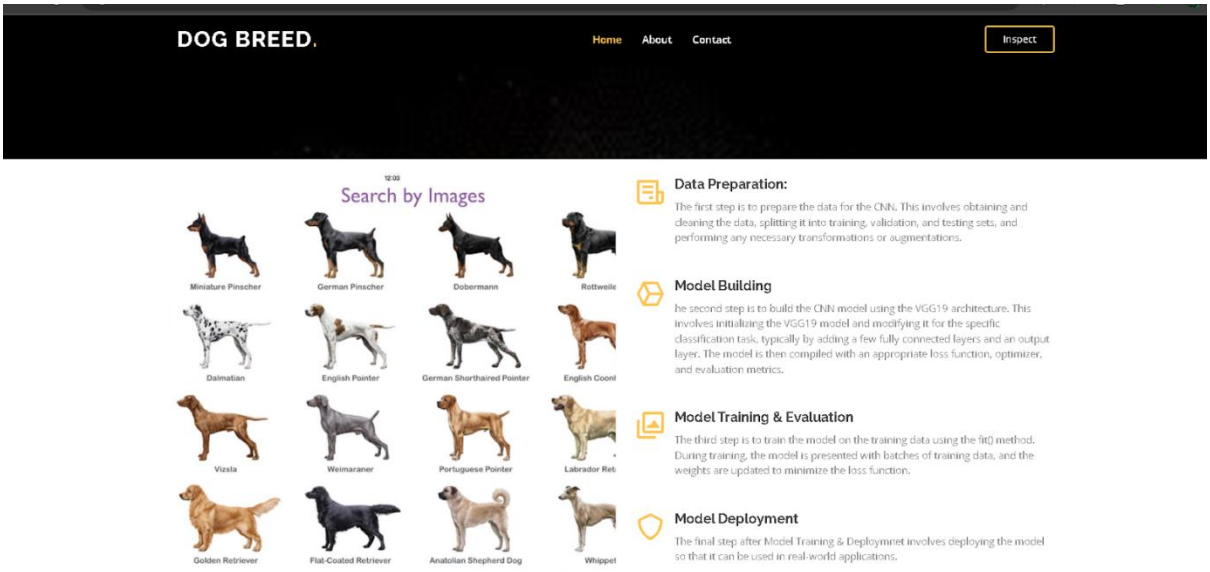
For more information regarding
HTML https://www.w3schools.com/html/

- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.
  - Link :CSS , JS

Home page:



About page:



Contact page:

Activity 2: Build Python code:

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 as argument Pickle library to load the model file.

app.py

```
1  import os
2  from flask import Flask, render_template, request, redirect, url_for, send_from_directory, flash
3  from werkzeug.utils import secure_filename
4  from utils import load_trained_model, predict_image
5
```

Creating our flask application and loading our model by using load_model method

```
9
10  os.makedirs(UPLOAD_FOLDER, exist_ok=True)
11
12  app = Flask(__name__)
13  app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
14  app.secret_key = 'replace-me-with-a-secure-key'
15
```

Activity 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_page():
    return render_template('predict.html')
```

Showcasing prediction on UI:

```
def result():
    if 'file' not in request.files:
    file = request.files['file']
    if file.filename == '':
        flash('No selected file')
        return redirect(request.url)
    if file and allowed_file(file.filename):
        filename = secure_filename(file.filename)
        # make unique
        import time
        filename = f"{int(time.time())}_{filename}"
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(filepath)

        # If model not loaded, show helpful message
        if MODEL is None or CLASS_MAP is None:
            flash('Model not found. Please run training first: python train_model.py')
            return redirect(url_for('predict_page'))

        try:
            label, conf = predict_image(filepath, MODEL, CLASS_MAP)
            confidence_pct = round(conf * 100, 2)
            return render_template('output.html', filename=filename, label=label, confidence=confidence_pct)
        except Exception as e:
            flash('Prediction failed: ' + str(e))
            return redirect(url_for('predict_page'))
    else:
        flash('Invalid file type. Allowed types: png, jpg, jpeg, gif')
        return redirect(request.url)
```

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.

```
@app.route('/uploads/<filename>')
def uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'], filename)


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

Activity 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.

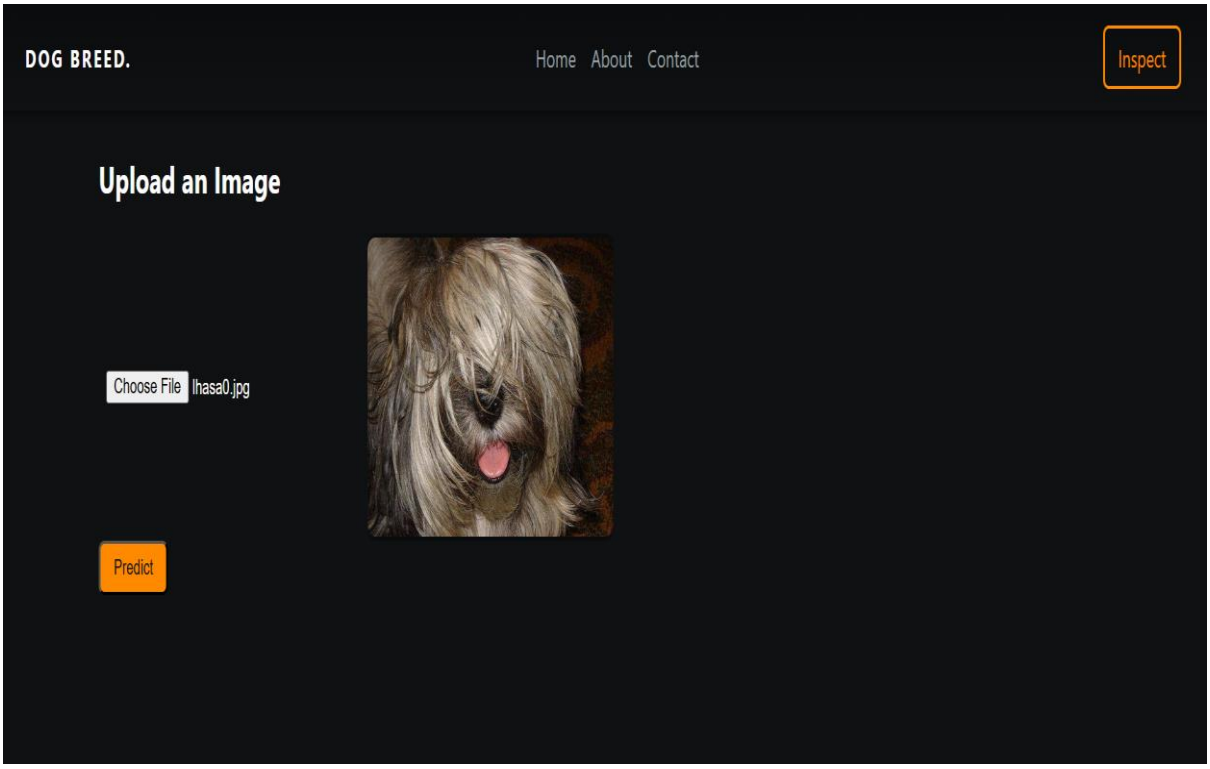
Then it will run on localhost: 5000

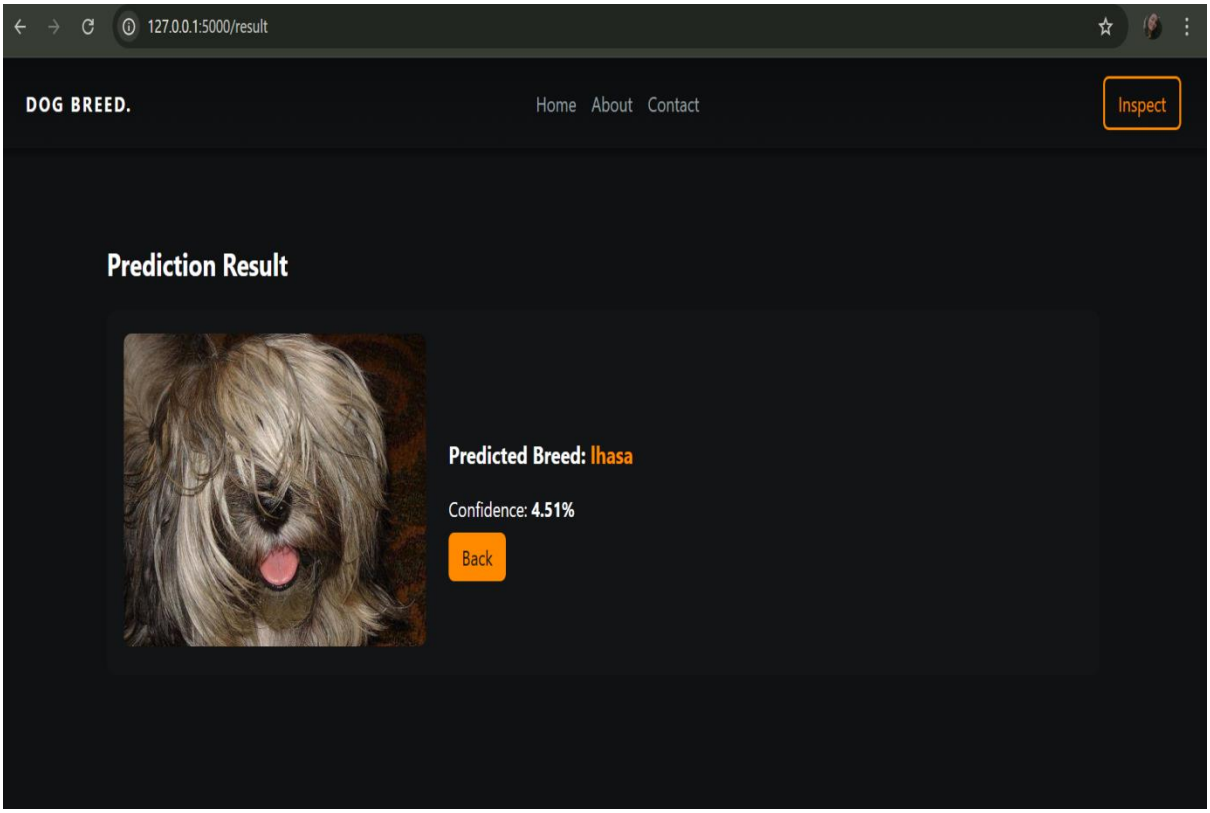
```
PS C:\ApssdcProject\DogBreedPrediction> python app.py

2026-02-19 15:16:08.082786: I tensorflow/core/util/port.cc:153] oneDNN custom operation
s are on. You may see slightly different numerical results due to floating-point round-
off errors from different computation orders. To turn them off, set the environment var
iable `TF_ENABLE_ONEDNN_OPTS=0`.
2026-02-19 15:16:10.693378: I tensorflow/core/util/port.cc:153] oneDNN custom operation
s are on. You may see slightly different numerical results due to floating-point round-
off errors from different computation orders. To turn them off, set the environment var
iable `TF_ENABLE_ONEDNN_OPTS=0`.
2026-02-19 15:16:18.884821: I tensorflow/core/platform/cpu_feature_guard.cc:210] This T
ensorFlow binary is optimized to use available CPU instructions in performance-critical
 operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow
 with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built.
`model.compile_metrics` will be empty until you train or evaluate the model.
 * Serving Flask app 'app'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production depl
oyment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with watchdog (windowsapi)
2026-02-19 15:16:20.016561: I tensorflow/core/util/port.cc:153] oneDNN custom operation
s are on. You may see slightly different numerical results due to floating-point round-
off errors from different computation orders. To turn them off, set the environment var
iable `TF_ENABLE_ONEDNN_OPTS=0`.
```

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

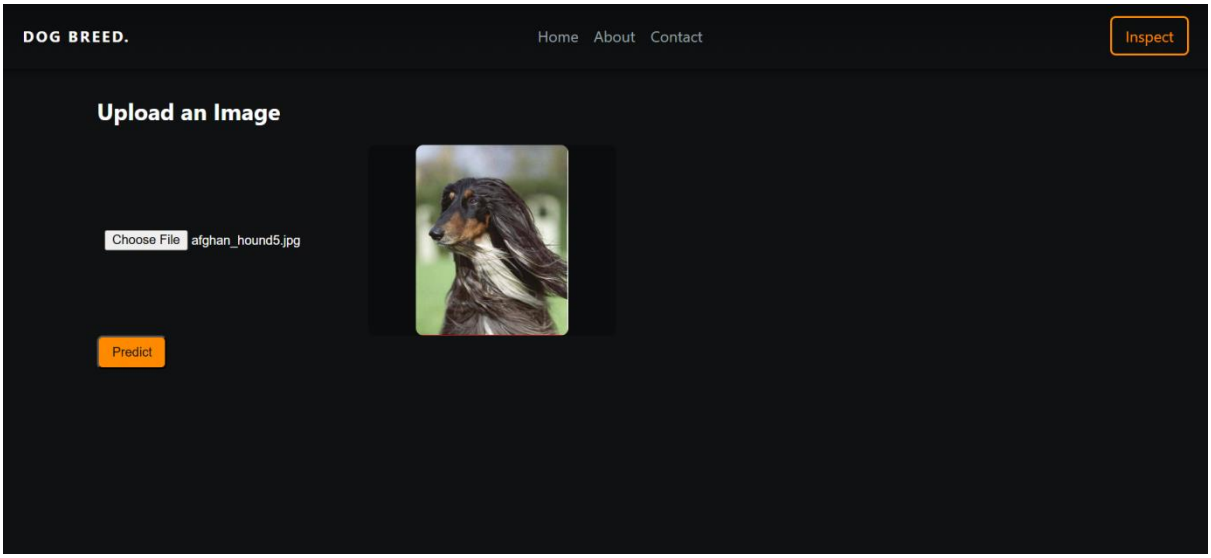
FINAL OUTPUT:

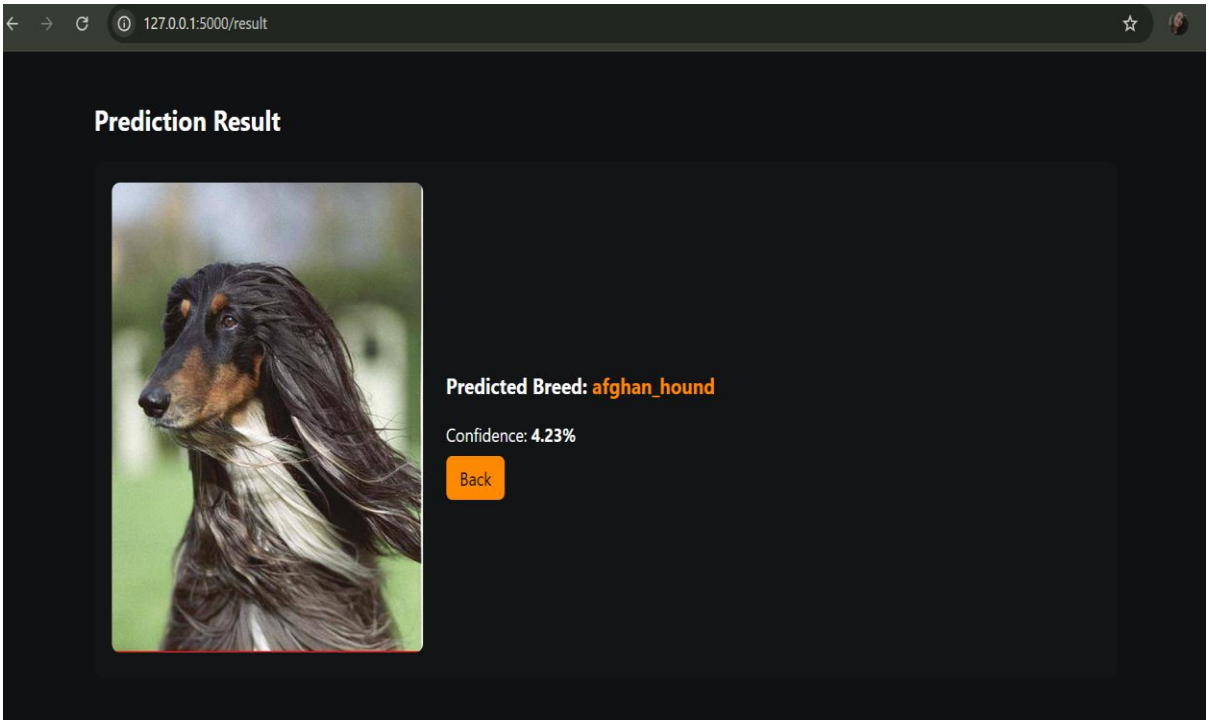
Output 1:


DOG BREED.
Home About Contact
Inspect
Upload an Image
Choose File
lhasa0.jpg
Predict


127.0.0.1:5000/result
DOG BREED.
Home About Contact
Inspect
Prediction Result
Predicted Breed: lhasa
Confidence: 4.51%
Back

Output 2:


DOG BREED.
Home
About
Contact
Inspect
Upload an Image
Choose File
afghan_hound5.jpg
Predict


127.0.0.1:5000/result
Prediction Result
Predicted Breed: afghan_hound
Confidence: 4.23%
Back