

Dog breed identification using transfer learning

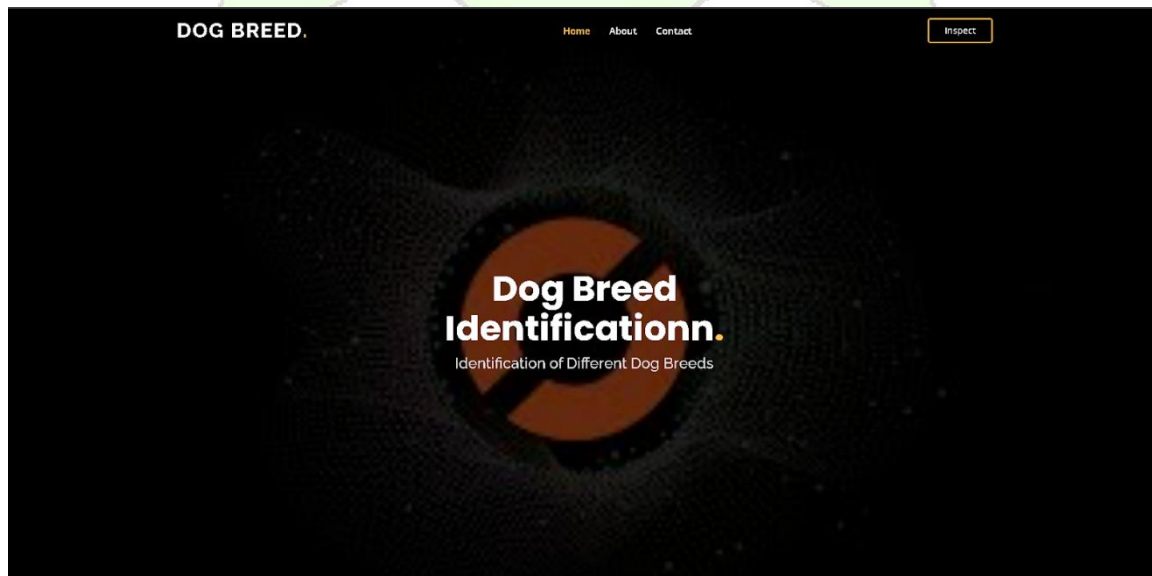
DATE	28-02-2026
TEAM ID	LTVIP2026TMIDS90703
PROJECT NAME	Dog breed identification using transfer learning
MAXIMUM MARKS	4 MARKS

6.5 - Application Building

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](#)

index.html looks like this




Dog breed identification using transfer learning


About Section: -


DOG BREED.[Home](#)[About](#)[Contact](#)[Inspect](#)


12/23


Search by Images



Miniature Pinscher



German Pinscher



Dobermann



Rottweiler



Dalmatian



English Pointer



German Shorthaired Pointer



English Cocker



Vizsla



Weimaraner



Portuguese Pointer



Labrador Retriever


Golden Retriever



Flat-Coated Retriever


Anatolian Shepherd Dog



Whippet

**Data Preparation:**


The first step is to prepare the data for the CNN. This involves obtaining and cleaning the data, splitting it into training, validation, and testing sets, and performing any necessary transformations or augmentations.

**Model Building**

The second step is to build the CNN model using the VGG19 architecture. This involves initializing the VGG19 model and modifying it for the specific classification task, typically by adding a few fully connected layers and an output layer. The model is then compiled with an appropriate loss function, optimizer, and evaluation metrics.

**Model Training & Evaluation**

The third step is to train the model on the training data using the fit() method. During training, the model is presented with batches of training data, and the weights are updated to minimize the loss function.


**Model Deployment**

The final step after Model Training & Deployment involves deploying the model so that it can be used in real-world applications.


Contact Us: -

CONTACT


CONTACT US

**Location:**

Survey no. 91, Sundarayya Vignana Kendram,
Technical Block, 6th floor, Madhava Reddy Colony,
Gachibowli, Hyderabad, Telangana 500032

**Email:**

info@thesmartbridge.com

**Call:**

+91 6304320044

Dog breed identification using transfer learning

2: Build python code

Building Python Code

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 our flask application and loading our model by using load_model method

```
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.

Dog breed identification using transfer learning

```
@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)
```

Dog breed identification using transfer learning

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.

