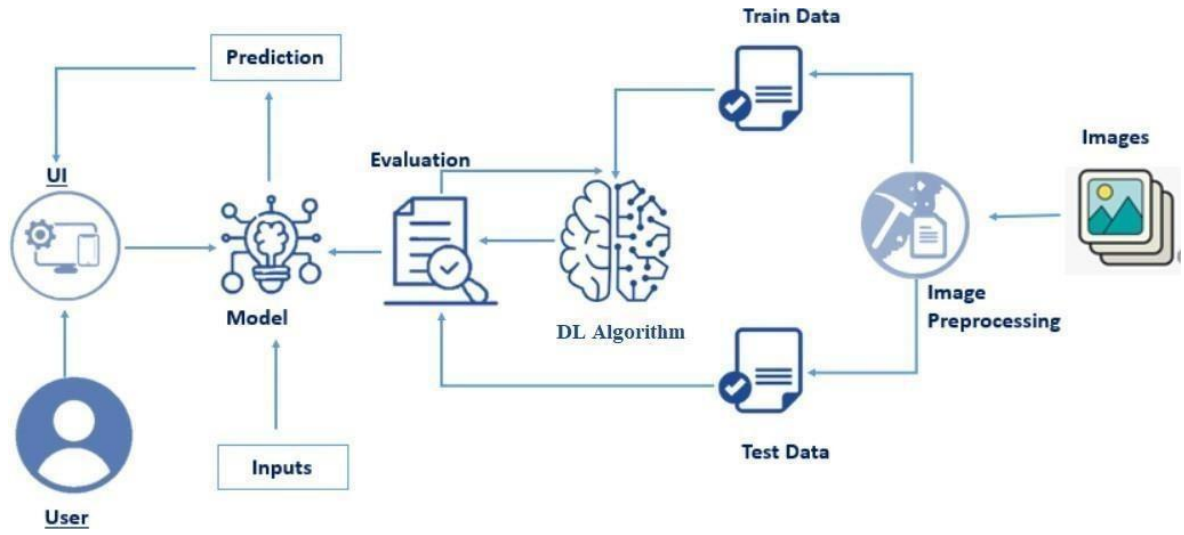
Dog Breed Identification using Transfer Learning

Introduction:

The field of dog breed identification involves training a machine learning model to classify dog images into specific breed categories. This task is made complex by the wide variety of dog breeds,each having distinct physical characteristics, such as size, shape, coat color, and facial features. Deep learning algorithms can learn to recognize these unique features and patterns in images to makeaccurate predictions about the breed of a given dog. Here we use Transfer Learning Approach i.e., VGG19 Architecture for dog breed identification, a large dataset of labeled dog images is required.

This dataset should include images from various dog breeds, with each image properly labeled with its corresponding breed. During the training phase, the Model learns to associate specific visual patterns and features with each breed, adjusting its internal weights to improve classification accuracy. Dog breed identification using machine learning has numerous practical applications. It can assist veterinarians in diagnosing and treating specific breeds, help dog owners better understand their pets, and aid in dog-related services such as adoption, breeding, and training. Furthermore, it can contribute to research efforts in studying the genetic and phenotypic characteristics of different dog breeds.

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, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code. For this project, we will be using Jupyter notebook and VS code.

To install Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video

Link: Click here to watch the video

1. To build Machine learning models you must require the following packages

- Numpy:
  - It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations

- Scikit-learn:

  - It is a free machine learning library for Python. It features various algorithms like supportvector machine, random forests, and k-neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy

- Flask:
  Web framework used for building Web applications

- Python packages:
  - open anaconda prompt as administrator
  - Type “pip install numpy” and click enter. o
    Type “pip install pandas” and click enter. o
    Type “pip install scikit-learn” and click enter.
  - Type “pip install tensorflow==2.12.0” and click enter.
    - Type “pip install keras==2.12.0” and click enter.
    - Type “pip install Flask” and click enter.
- Deep Learning Concepts

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

  - VGG19: VGG19 is a deep convolutional neural network architecture for image classification, consisting of 19 layers with small convolution filters.
    VGG19

  - Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications. Flask Basics

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

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:

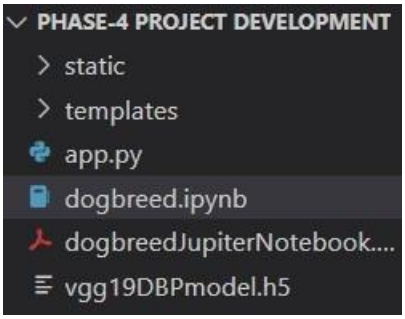
- The user interacts with the UI (User Interface) to choose the image.
- The chosen image is analyzed by the model which is integrated with flask application.
- CNN Models analyze the image, then prediction is showcased on the FlaskUI.

To accomplish this, we must complete all the activities and tasks listed below.

- o Data Collection. o Create Train and Test Folders.
- o Data Preprocessing.
- o Import the ImageDataGenerator library o Configure ImageDataGenerator class
- o ApplyImageDataGenerator functionality to Train dataset and Test dataset
- o Model Building o Import the model building Libraries.
- o Importing the VGG19. o Initializing the model o Adding Fully connected Layer oConfigure the Learning Process o Training and Testing the model.
- o Save the Model
- o Application Building o Create an HTML file o Build Python Code

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 vgg19DBPmodel.h5 ● Templates folder contains index.html & other html pages. ● The Static folder contains css & js files.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem.

Refer Project Description

Activity 2: Business requirements

Here are some potential business requirements for Dog Breed Identification.

a. Accurate Prediction:
The predictor must be able to accurately classify the images of different Dog Breeds. So that there will be no misclassification which decreases the accuracy of the model.

b. Real-time data acquisition:
The predictor must be able to acquire real-time data from the various sources. The data acquisition must be seamless and efficient to ensure that the predictor is always up-to-date with the latest information.

c. User-friendly interface:
The predictor must have a user-friendly interface that is easy to navigate and understand. The interface should present the results of the predictor in a clear and concise manner. d.

Report generation:

Generate a report outlining the predicted Dog Breeds. By analyzing data and applying advanced algorithms, the project generates detailed reports that include key findings, Dog Breed classifications, and relevant insights. The report should be presented in a clear and concise manner, with appropriate insights to help patients confirm their results.

Activity 3: Literature Survey (Student Will Write)

A literature survey would involve researching and reviewing existing studies, articles, and other publications on the topic of the project. The survey would aim to gather information on current systems, their strengths and weaknesses, and any gaps in knowledge that the project could address. The literature survey would also look at the methods and techniques used in previous projects, and any relevant data or findings that could inform the design and implementation of the current project.

Activity 4: Social or Business Impact.

The Dog Breed Identification project can have both social and business impacts.

Social Impact:

Accurate dog breed identification contributes to improved dog welfare. By understanding the breed's unique needs, such as exercise requirements, grooming needs, and potential health risks, owners can provide appropriate care and ensure the well-being of their dogs. Dog breed identification promotes responsible pet ownership. Understanding the specific breed characteristics, temperament, and exercise requirements helps potential dog owners make informed decisions about which breed is best suited to their lifestyle, living situation, and family dynamics.

Business Impact:

Pet stores and breeders can leverage dog breed identification to provide accurate information about the breeds they offer. This helps potential customers make informed decisions when choosing adog based on their lifestyle, preferences, and compatibility. Dog breed identification can influence thedevelopment of pet services and products. Understanding specific breed characteristics allows pet service providers, such as trainers, groomers, and veterinarians, to tailor their services to meet the unique needs of different breeds. It can also inform the design and marketing of products, such as breed-specific food, toys, and accessories.

Milestone 2: 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 collectedimages of 120 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 distortionsor enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

Importing Libraries.

```
[ ]: #importing all the libraries
import os
import numpy as np
import pandas as pd
from keras.utils import np_utils
from sklearn.preprocessing import LabelEncoder
from keras.applications.resnet import preprocess_input
from tensorflow.keras.utils import load_img
from tensorflow.keras.utils import img_to_array

import matplotlib.pyplot as plt
%matplotlib inline
```

```
[ ]: from keras.callbacks import EarlyStopping
from keras.backend import clear_session

clear_session()
```

This section imports necessary libraries for data manipulation, image preprocessing, and model building. It includes tools for working with images, dataframes, and deep learning with Keras.

Setting up the Data.

```
[ ]: #train dir contains the training images
base_dir = '.'
data_dir = os.path.join(base_dir,'train')
files = os.listdir(data_dir)
```

```
[ ]: #target information from labels.csv

labels = pd.read_csv(os.path.join(base_dir,'labels.csv'))
labels.head()
```

[ ]:

	id	breed
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull
1	001513dfcb2ffafc82cccf4d8bbaba97	dingo
2	001cdf01b096e06d78e9e5112d419397	pekinese
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick
4	0021f9ceb3235effd7fcde7f7538ed62	golden_retriever

[ ]:
```
file_df = pd.DataFrame({'id':list(map(lambda x:x.replace('.jpg',''),files))})
file_df.head()
```

[ ]:

	id
0	000bec180eb18c7604dcecc8fe0dba07
1	001513dfcb2ffafc82cccf4d8bbaba97
2	001cdf01b096e06d78e9e5112d419397
3	00214f311d5d2247d5dfe4fe24b2303d
4	0021f9ceb3235effd7fcde7f7538ed62

[ ]:
```
#mapping file with breed, maintain file read order

label_info =  pd.merge(left = file_df, right = labels)
label_info.head()
```

[ ]:

	id	breed
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull
1	001513dfcb2ffafc82cccf4d8bbaba97	dingo
2	001cdf01b096e06d78e9e5112d419397	pekinese
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick
4	0021f9ceb3235effd7fcde7f7538ed62	golden_retriever

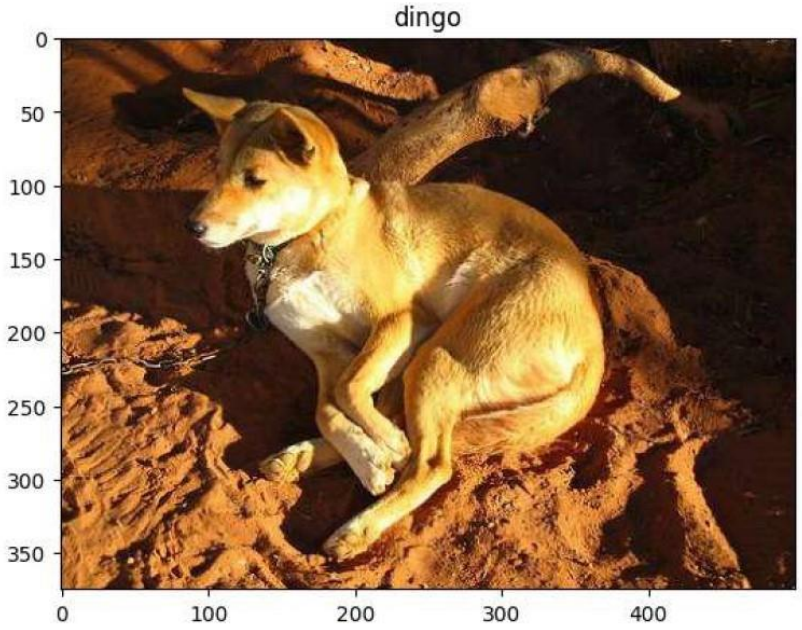
This part organizes the training image data by reading file names, loading target information from 'labels.csv,' and creating a DataFrame ('label_info') that associates file names with corresponding labels

Preprocessing and Visualization.

```
[ ]: img = plt.imread(os.path.join(data_dir,files[1]))
```

```
[ ]: #showing a image

plt.imshow(img)
plt.title(label_info.iloc[1]['breed'])
plt.show()
```



Here, an image is loaded and displayed for visualization purposes

Label Encoding

```
[ ]: # converting target to one hot vector format

num_classes = len(label_info.breed.unique())
num_classes
```

```
[ ]: 120
```

```
[ ]: le = LabelEncoder()
breed = le.fit_transform(label_info.breed)
Y = np_utils.to_categorical(breed,num_classes = num_classes)
```

```
[ ]: Y.shape
```

```
[ ]: (10222, 120)
```

This section encodes the dog breed labels into numerical format and converts them to one-hot vectors.

Image Loading and Preprocessing

```
[ ]: # converting image to numpy array
     input_dim = (224, 224)

     X = np.zeros((Y.shape[0], *input_dim,3))

     for i,img in enumerate(files):
         image = load_img(os.path.join(data_dir,img), target_size = input_dim)
         image = img_to_array(image)
         image = image.reshape((1, *image.shape))
         image = preprocess_input(image)
         X[i] = image
```

```
[ ]: X.shape
```

```
[ ]: (10222, 224, 224, 3)
```

Images are loaded, resized to (224, 224), and preprocessed using the VGG19-specific preprocessing function.

Building the VGG19 Model

```
[ ]: from keras.applications.vgg19 import VGG19
     from keras.models import Model
     from keras.layers import Dense,GlobalAveragePooling2D, Flatten, Dropout

     vgg_model = VGG19(weights='imagenet', include_top=False)

     x= vgg_model.output
     x= GlobalAveragePooling2D()(x)
     x=Dropout(0.3)(x)
     out = Dense(120,activation = 'softmax')(x)

     model = Model(inputs=vgg_model.input, outputs=out)
```

This section initializes the VGG19 model, removes the top layer, adds a global average pooling layer, dropout layer, and a dense output layer for 120 dog breeds.

Freezing Layers

```
for layer in vgg_model.layers[:-1]:
    layer.trainable = False
for layer in vgg_model.layers[-1:]:
```

This code freezes all layers in the VGG19 model except for the last layer.

Model Compilation

```
    layer.trainabl= True
from keras.optimizers import Adam
opt= Adam()

model.compile(optimizer = opt, loss = 'categorical_crossentropy', metrics = ['accuracy'])
model.summary()
```

This part configures the model for training using the Adam optimizer and categorical crossentropy loss.

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, None, None, 3)]	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1792
block1_conv2 (Conv2D)	(None, None, None, 64)	36928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73856
block2_conv2 (Conv2D)	(None, None, None, 128)	147584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295168
block3_conv2 (Conv2D)	(None, None, None, 256)	590080
block3_conv3 (Conv2D)	(None, None, None, 256)	590080
block3_conv4 (Conv2D)	(None, None, None, 256)	590080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1180160
block4_conv2 (Conv2D)	(None, None, None, 512)	2359808
block4_conv3 (Conv2D)	(None, None, None, 512)	2359808
block4_conv4 (Conv2D)	(None, None, None, 512)	2359808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2359808
block5_conv2 (Conv2D)	(None, None, None, 512)	2359808
block5_conv3 (Conv2D)	(None, None, None, 512)	2359808
block5_conv4 (Conv2D)	(None, None, None, 512)	2359808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
dropout (Dropout)	(None, 512)	0
dense (Dense)	(None, 120)	61560

Total params: 20,085,944
Trainable params: 61,560
Non-trainable params: 20,024,384

Model Training

```
history_few_layer = model.fit(X[:1000], Y[:1000], batch_size=32, epochs=30, validation_split=0.2, verbose=2)
#history_few_layer = model.fit(X[:1000], Y[:1000], batch_size=32, epochs=30, validation_split=0.2, verbose=2, callbacks='earlystop')
```

The model is trained on the first 1000 samples for 30 epochs with a batch size of 32 and a validation split of 20%.

```
history_few_layer = model.fit(X, Y, batch_size=32, epochs=30, validation_split=0.2, verbose=2)
#history_few_layer = model.fit(X[:1000], Y[:1000], batch_size=32, epochs=30, validation_split=0.2, verbose=2, callbacks='earlystop')
```

The model is trained on all samples for 30 epochs with a batch size of 32 and a validation split of 20%

Model Saving

```
[ ]: model.save('vgg19DBPmodel.h5')
```

The trained model is saved in an H5 file.

Model Evaluation

```
[ ]: history_few_layer.history.keys()

[ ]: dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

This line retrieves keys from the training history, which can be used for further analysis or visualization.

Milestone 4: Application Building

Now that we have trained our model, let us build our flask application which will be running inour local browser with a user interface.

In the flask application, the input parameters are taken from the HTML page These factors arethen given to the model to know to predict the type of Colon Diseases and showcased on the

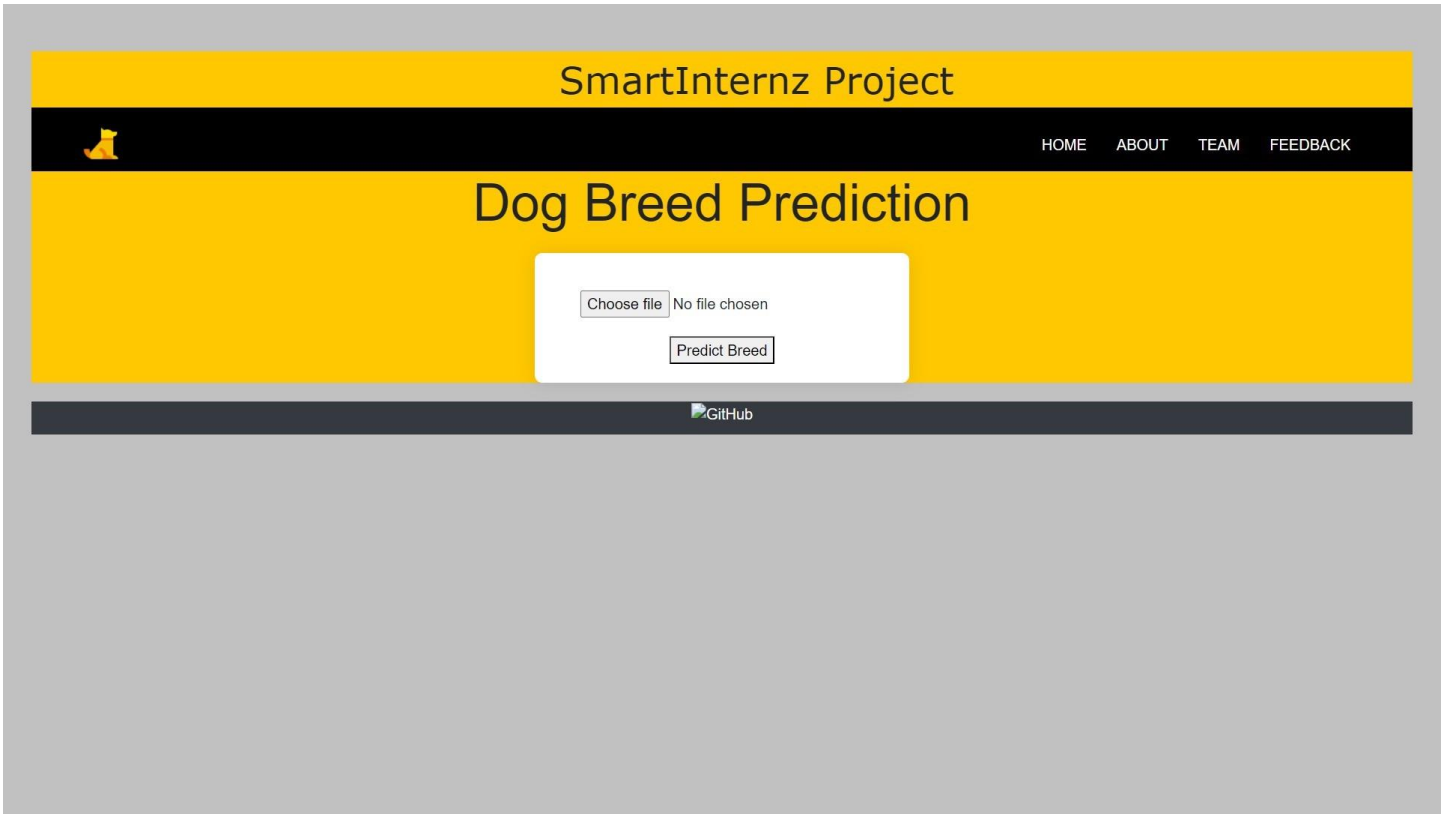
HTML page to notify the user. Whenever the user interacts with the UI and selects the “Inspect” button, the next page is opened where the user chooses the image and predicts the output.

Activity 1 : Create HTML Pages o We use HTML to create the front end part of the web page.

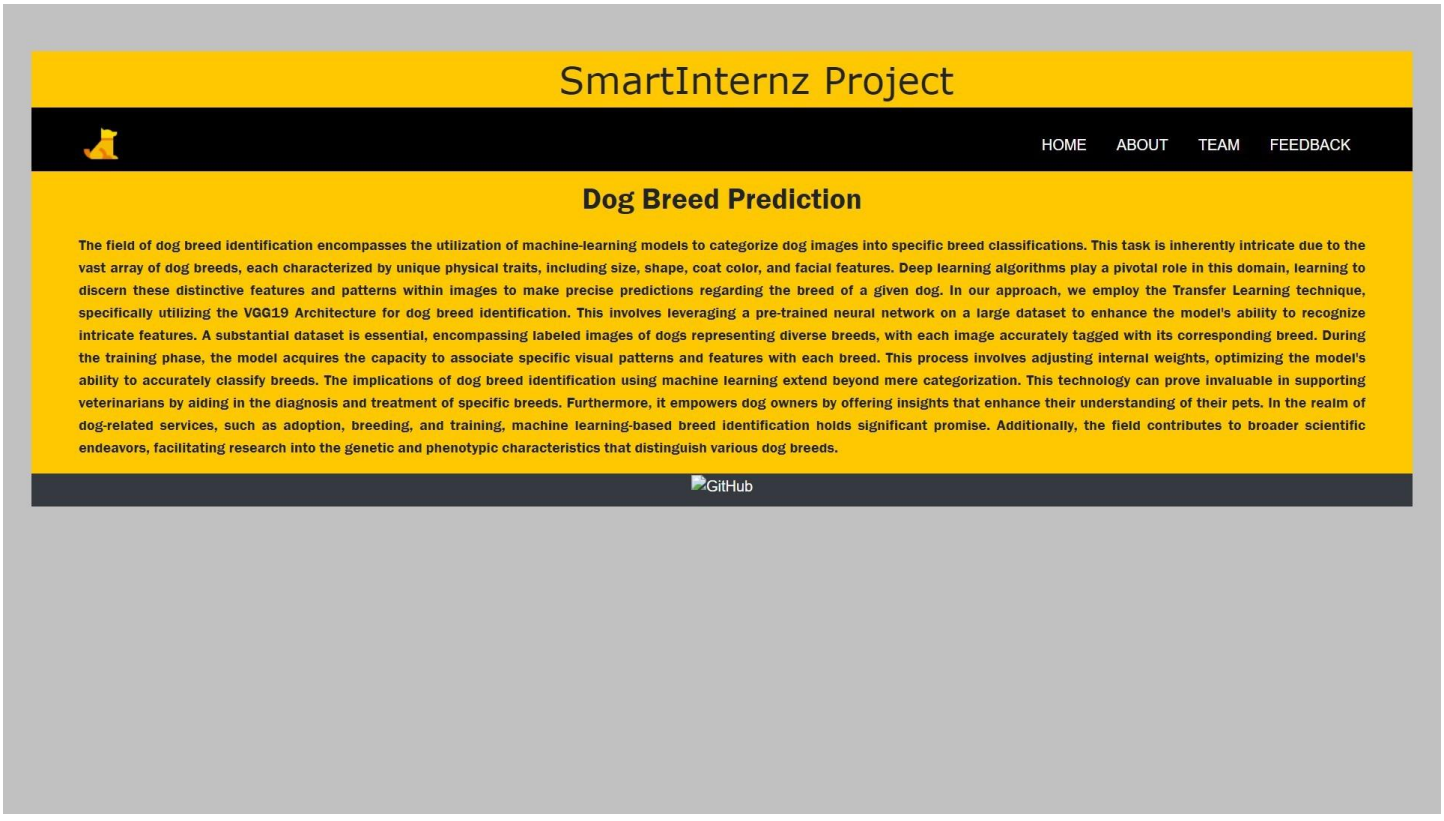
- Here, we have created 3 HTML pages- index.html, about.html, feedback.html and team.html o index.html displays the home page. o about.html displays an introduction about the project. o feedback.html helps us for development. o team.html displays the team members.
- For more information regarding HTML https://www.w3schools.com/html/
- We also use JavaScript-main.js and CSS-main.css to enhance our functionalityand view of HTML pages.

○ Link :CSS , JS

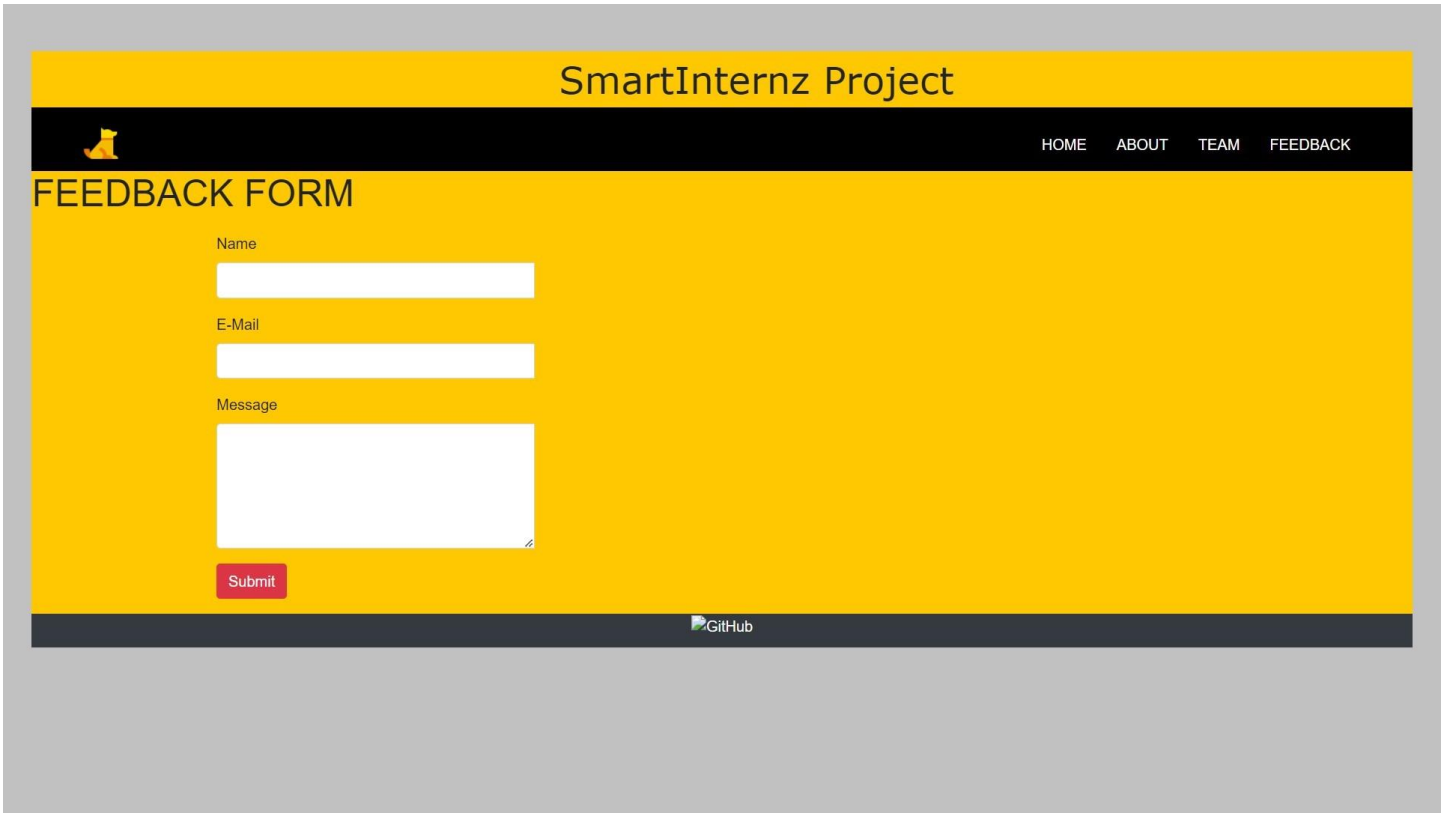
index.html looks like this



about.html:-



feedback.html:-



Activity 2: Build python code

Task 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.

```
from flask import Flask, render_template, request
from PIL import Image
import numpy as np
from tensorflow.keras.applications.vgg19 import preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg19 import VGG19
import os
from io import BytesIO
```

Task 2: Creating our flask application and loading our model by using load_model method

```
app = Flask(__name__)
model = VGG19(weights='imagenet')
#model = load_model('vgg19DBPmodel.h5')
```

Task 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('/feedback')
def feedback():
    return render_template('feedback.html')

@app.route('/team')
def team():
    return render_template('team.html')

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

```
@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return render_template('index.html', prediction="No file selected!")

    file = request.files['file']

    if file.filename == '':
        return render_template('index.html', prediction="No file selected!")
```

Showcasing prediction on UI:

```
img = Image.open(file)
img = img.resize((224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array = preprocess_input(img_array)

predictions = model.predict(img_array)
decoded_predictions = decode_predictions(predictions, top=1)[0][0]

breed_prediction = decoded_predictions[1]

# Replace this with your actual class mapping
#class_mapping = {
    #0: 'boston_bull', 1: 'dingo', 2: 'pekinese', 3: 'bluetick', 4: 'golden_retriever', 5: '
    #10: 'walker_hound', 11: 'maltese_dog', 12: 'norfolk_terrier', 13: 'african_hunting_dog'
    #20: 'standard_schnauzer', 21: 'irish_water_spaniel', 22: 'black-and-tan_coonhound', 23:
    #30: 'groenendael', 31: 'dhole', 32: 'toy_poodle', 33: 'border_terrier', 34: 'tibetan_te
    #40: 'greater_swiss_mountain_dog', 41: 'basset', 42: 'australian_terrier', 43: 'schipperk
    #50: 'brittany_spaniel', 51: 'kelpie', 52: 'papillon', 53: 'border_collie', 54: 'entlebu
    #60: 'pug', 61: 'malinois', 62: 'komondor', 63: 'airedale', 64: 'leonberg', 65: 'mexican_
    #70: 'cardigan', 71: 'italian_greyhound', 72: 'clumber', 73: 'scotch_terrier', 74: 'afgha
    #80: 'brabancon_griffon', 81: 'toy_terrier', 82: 'chow', 83: 'flat-coated_retriever', 84
    #90: 'newfoundland', 91: 'briard', 92: 'chesapeake_bay_retriever', 93: 'dandie_dinmont',
    #100: 'sealyham_terrier', 101: 'standard_poodle', 102: 'keeshond', 103: 'japanese_spanie
    #110: 'blenheim_spaniel', 111: 'silky_terrier', 112: 'sussex_spaniel', 113: 'german_short
#}

#breed_prediction = class_mapping.get(predicted_class_index, "Unknown")

temp_img_path = os.path.join('static', 'temp_img.jpg')
img.save(temp_img_path)
return render_template('index.html', prediction=breed_prediction, image_path=temp_img_path)
```

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 directoryusing 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 numericalvalue 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=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.

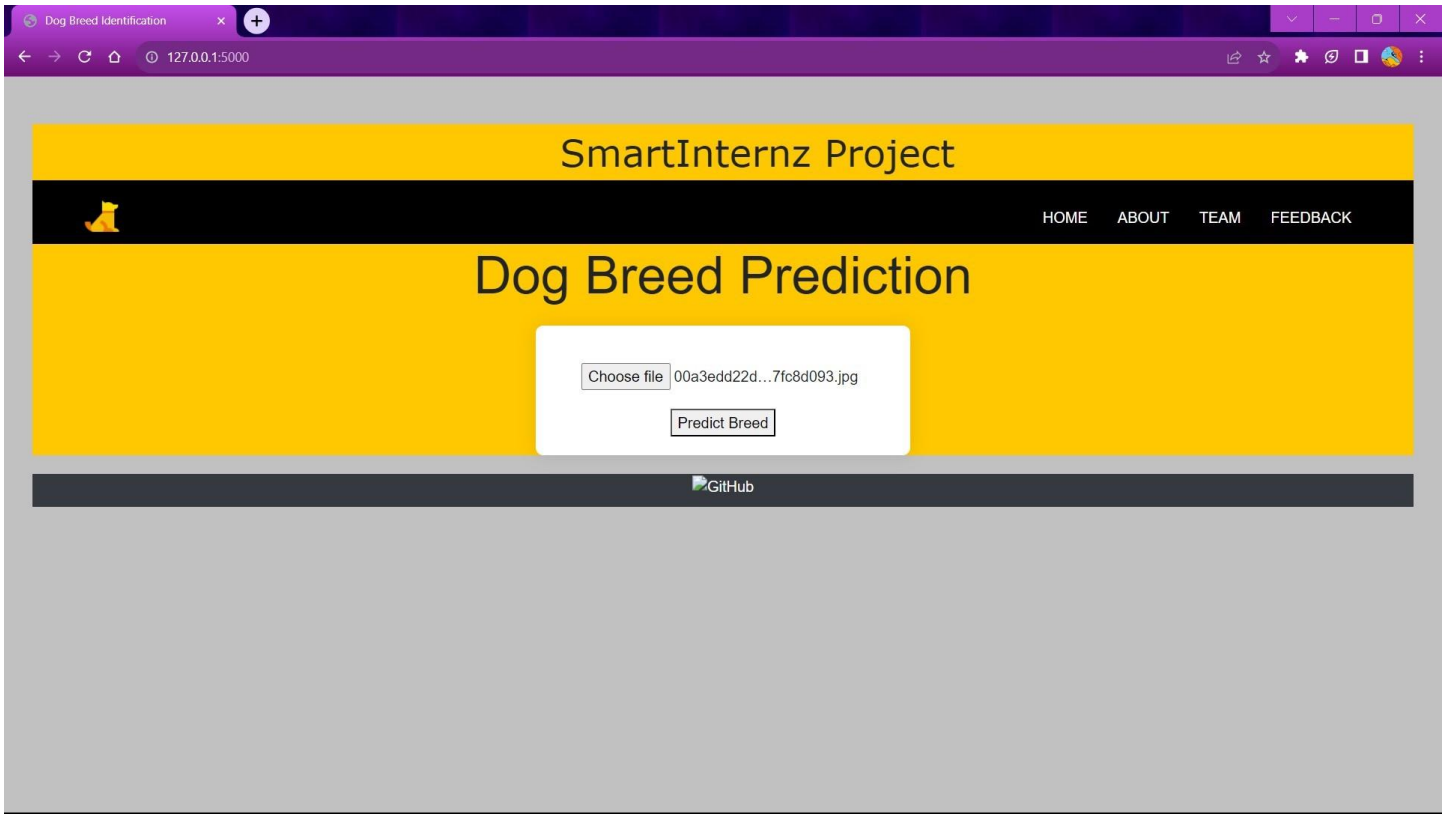
Then it will run on localhost: 5000

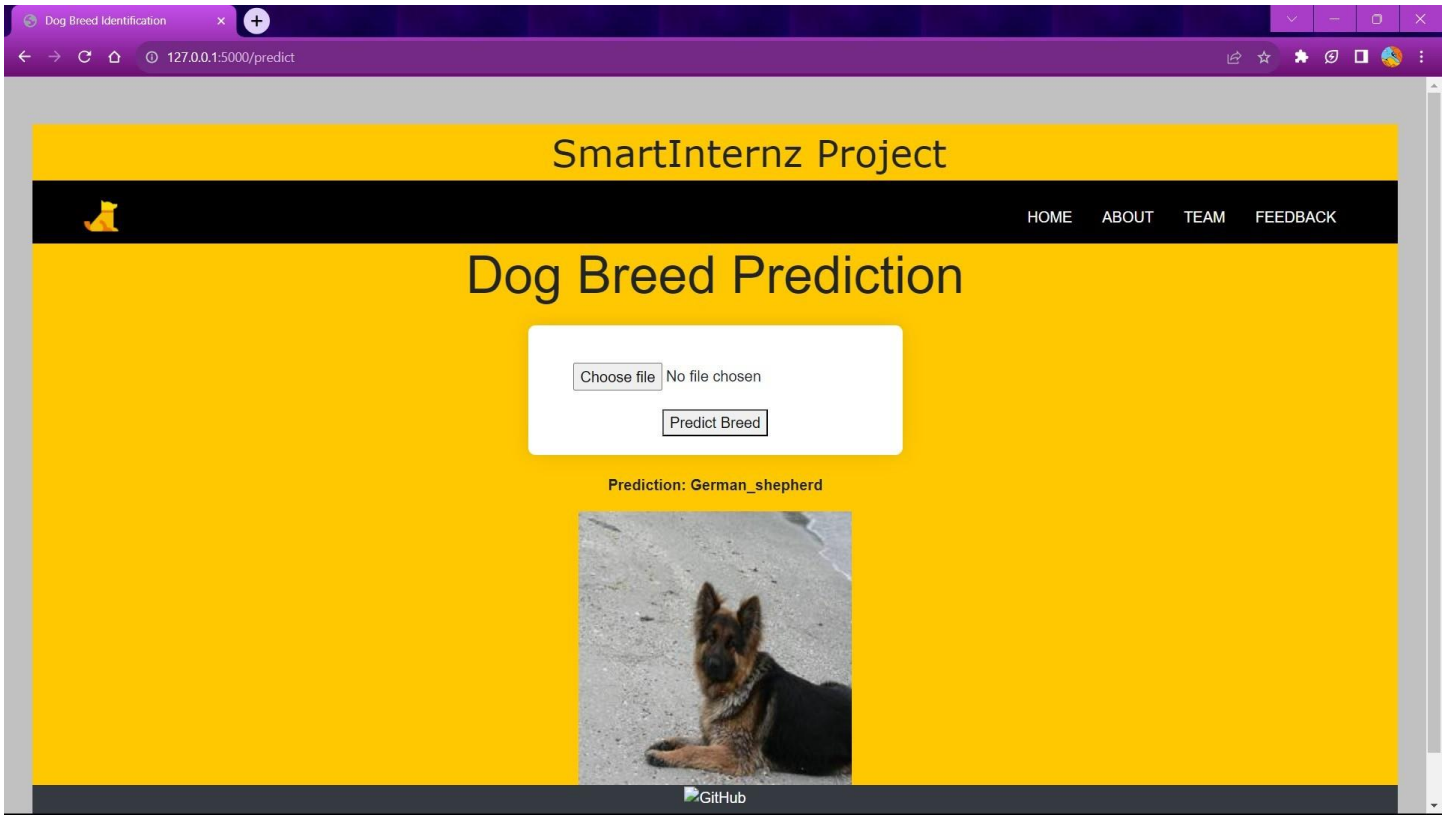
```
PS F:\Smart_Internz\Dog_Breed_prediction> python -u "f:\Smart_Internz\Dog_Breed_prediction\app.py"
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug 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.

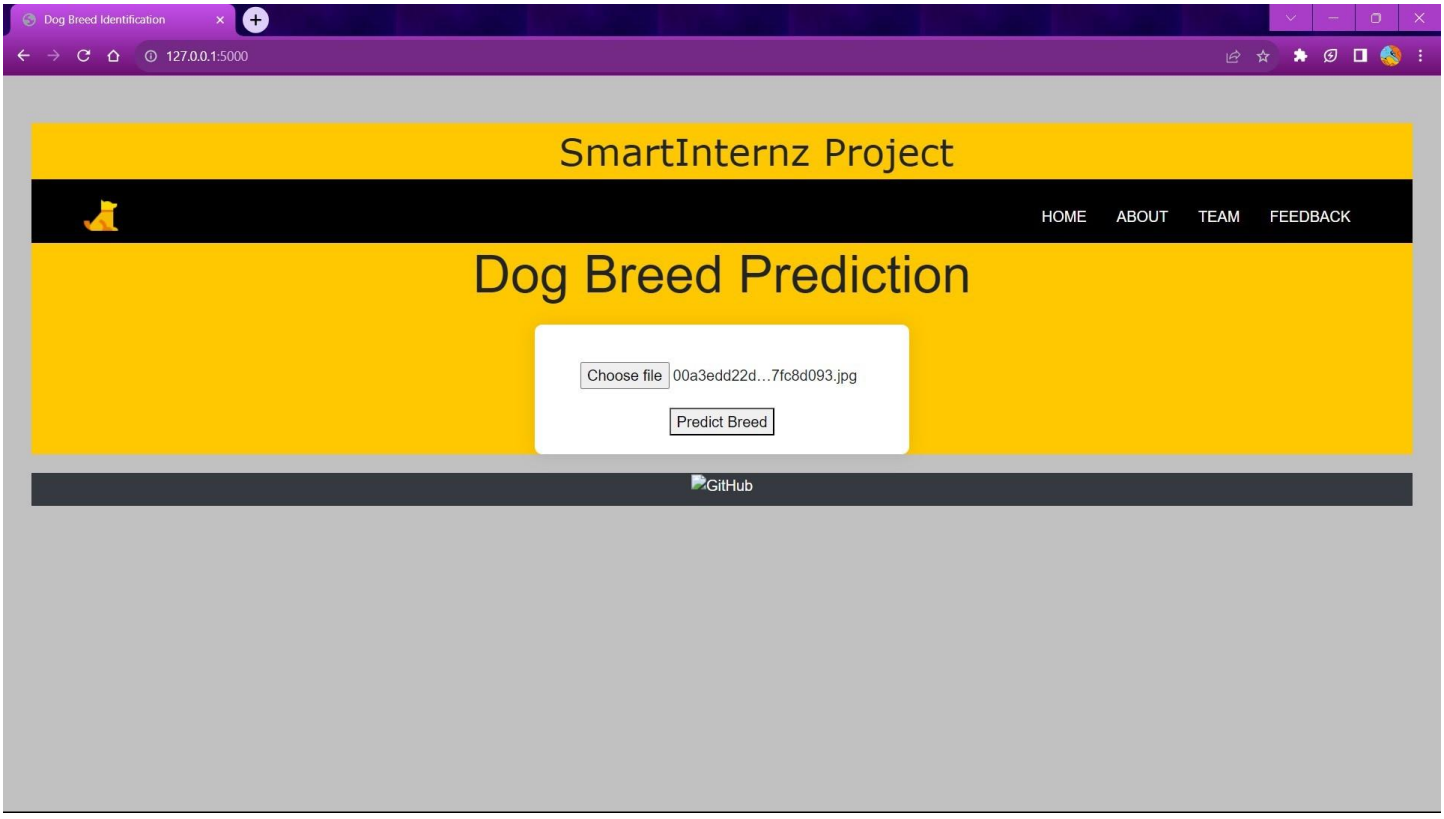
FINAL OUTPUT:

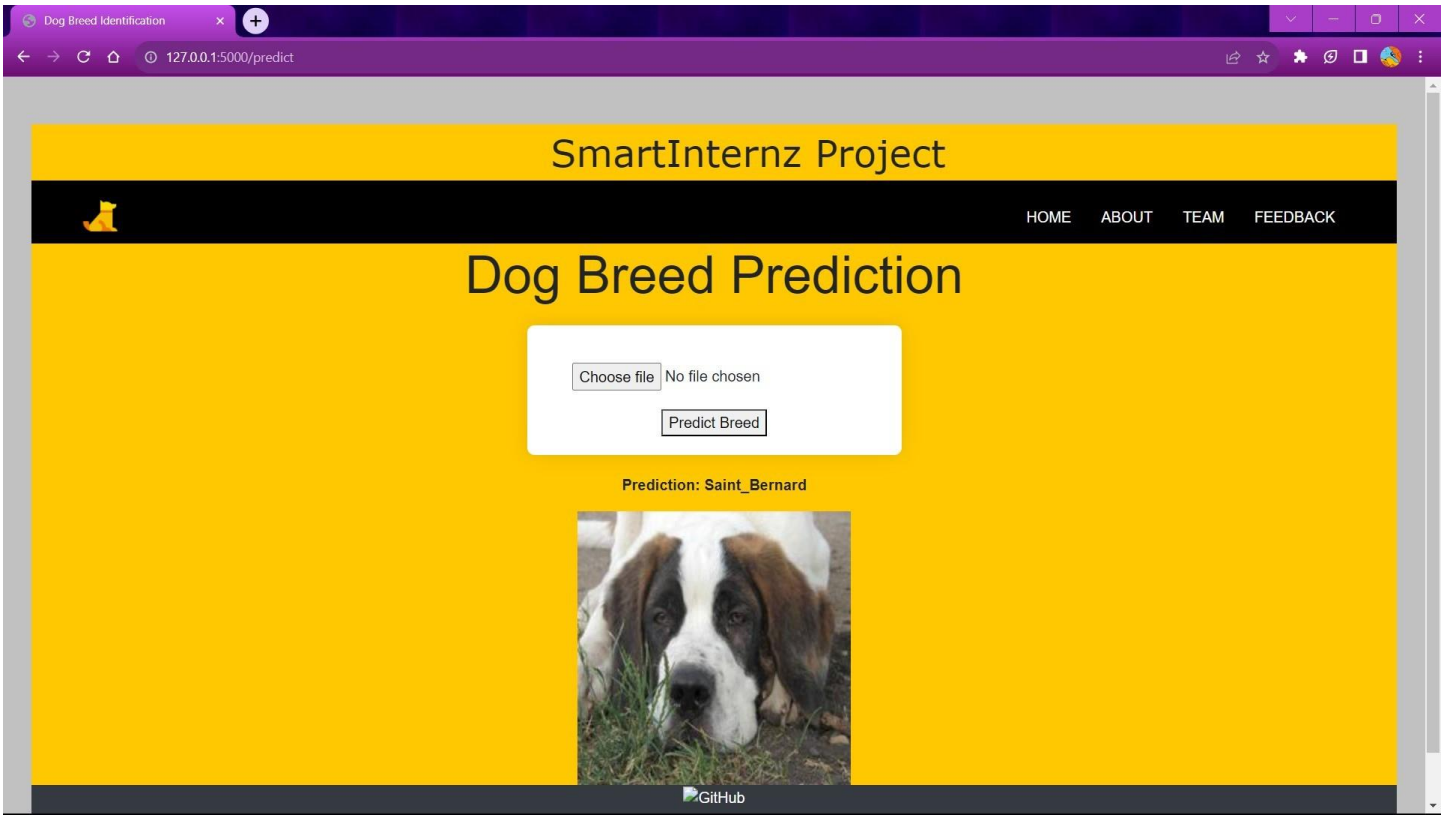
Output 1:




Dog Breed Identification
127.0.0.1:5000/predict
SmartInternz Project
HOME
ABOUT
TEAM
FEEDBACK
Dog Breed Prediction
Choose file
No file chosen
Predict Breed
Prediction: German_shepherd
GitHub

Output 2:


Dog Breed Identification
127.0.0.1:5000
SmartInternz Project
HOME
ABOUT
TEAM
FEEDBACK
Dog Breed Prediction
Choose file
00a3edd22d...7fc8d093.jpg
Predict Breed
GitHub


Dog Breed Identification
127.0.0.1:5000/predict
SmartInternz Project
HOME
ABOUT
TEAM
FEEDBACK
Dog Breed Prediction
Choose file
No file chosen
Predict Breed
Prediction: Saint_Bernard
GitHub