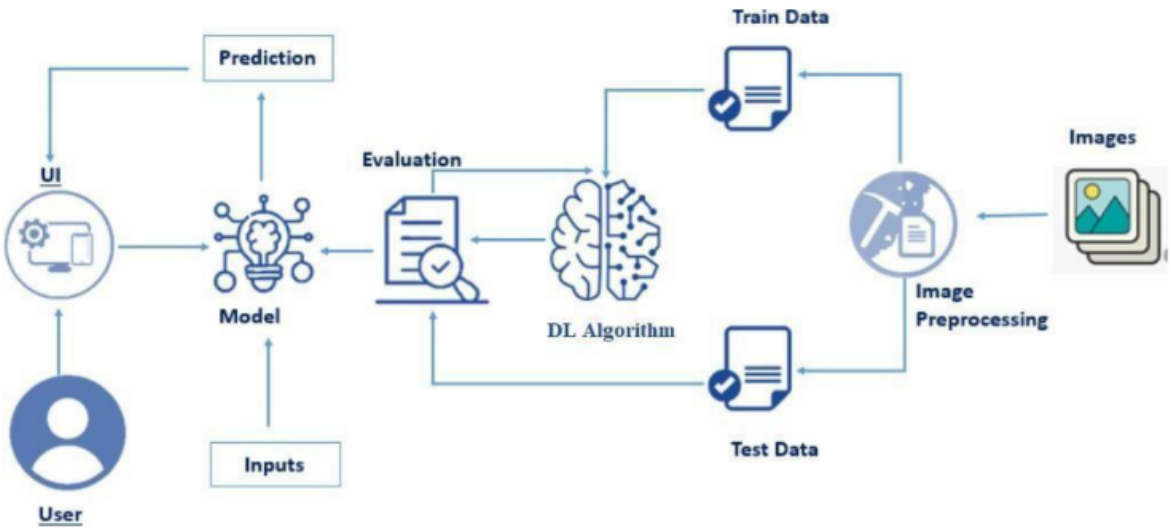
Dog Breed Identification Using Transfer Learning

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 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 make accurate predictions about the breed of a given dog. Here we use Transfer Learning Approach i.e.NASNetLarge 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

For this project, we will be using google colab and VS code.

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

● **Numpy:**

o 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:**

o It is a free machine learning library for Python. It features various algorithms like support vector 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:**

o open anaconda prompt as administrator
o Type "pip install numpy" and click enter.
o Type "pip install pandas" and click enter.
o Type "pip install scikit-learn" and click enter.
o Type "pip install tensorflow==2.12.0" and click enter.
o Type "pip install keras==2.12.0" and click enter.
o Type "pip install Flask" and click enter.

● **Deep Learning Concepts**

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

- **NASnetLarge:**

NASNet (Neural Architecture Search Network) is a type of deep learning model designed for image classification tasks. It was developed by Google's DeepMind in collaboration with Google Brain. NASNet utilizes a technique called neural architecture search (NAS) to automatically discover effective neural network architectures for a given task.

The "Large" in NASNet Large refers to a specific variant of the model that has a larger number of parameters compared to its smaller counterparts. A model with more parameters generally has a higher capacity to learn complex patterns, which can potentially lead to improved performance on challenging tasks.

o **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.

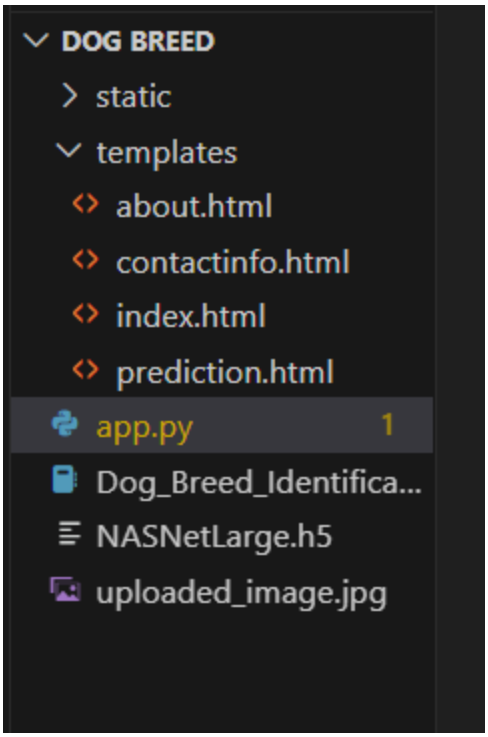
To accomplish this, we must complete all the activities and tasks listed below.

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 NASNetLarge.
o Initializing the model
o Adding Fully connected Layer
o Configure 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 **NasnetLarge.h5**
● templates folder contains **index.html**, **contact info.html** & **about.html pages,dog breed.png**.

<u>Milestone 1:</u> Define Problem / Problem Understanding

Activity 1: Specify the business problem.

The objective of this project is to develop a robust and accurate dog breed identification system using the technique of transfer learning. Transfer learning involves leveraging a pretrained deep learning model and finetuning it on a specific task, in this case, identifying dog breeds

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

"Fine-Grained Dog Breed Classification" by Z. Cao et al. (2017):

This paper introduces a fine-grained dog breed classification dataset called Stanford Dogs. It explores different deep learning architectures including NASNetLarge for this task.

"A Study of the Effects of Fine-Tuning on CNN-based Dog Breed Identification" by J. Gadea et al. (2018):

This study investigates the effects of fine-tuning a pre-trained CNN (including

NASNetLarge) for the specific task of dog breed identification.

"Improved Microorganism Identification Using Deep Learning Architectures" by A. Alferez et al. (2019):

While not specific to dogs, this paper explores the use of NASNetLarge and other models for microorganism identification and classification, which shares similarities with fine-grained classification tasks.

"Applying Transfer Learning to Image-based Plant Disease Identification" by H. Zhang et al. (2019):

This paper discusses the use of transfer learning with deep learning models like NASNetLarge for plant disease identification, which involves fine-grained classification similar to dog breed identification.

"Neural Architecture Search for Lightweight Non-English Speaker Identification System" by L. Yao et al. (2020):

While not directly related to dog breed identification, this paper explores the use of NAS for designing efficient models for speaker identification, showcasing the versatility of NAS in various classification tasks.

"Automated Identification of Street Pavement Distresses Using a Deep Learning Approach" by X. Xie et al. (2021):

This paper focuses on using deep learning models for identifying street pavement distresses, but it provides insights into the potential applications of NASNetLarge in fine-grained classification tasks.

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 a dog based on their lifestyle, preferences, and compatibility. Dog breed identification can influence the development 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.

<u>Milestone 2:</u> 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 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 distortions
or enhances some image features important for further processing, although perform some
geometric transformations of images like rotation, scaling, translation, etc.

Activity 1: Organizing the Images into Different Classes.

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

```
[7] import pandas as pd
    import numpy as np

    dataset_dir="/content/train"
    labels=pd.read_csv("/content/labels.csv")
    labels.head()
```

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

```
[8]  def make_dir(x):
       if os.path.exists(x)==False:
         os.makedirs(x)
     base_dir="./subset"
     make_dir(base_dir)
```

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

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

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.

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.

```
[11] 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.

```
[11] generator=ImageDataGenerator().flow_from_directory("/content/subset/train")

     Found 10222 images belonging to 120 classes.
```

Checking the dataset is balanced or imbalanced:

```
[12] import os

     data_dir = '/content/subset/train'
     classes = os.listdir(data_dir)

     for cls in classes:
         cls_path = os.path.join(data_dir, cls)
         num_samples = len(os.listdir(cls_path))
         print(f'Class {cls} has {num_samples} samples.')
```

```
Class dhole has 76 samples.
Class australian_terrier has 102 samples.
Class german_short-haired_pointer has 75 samples.
Class kuvasz has 71 samples.
Class basenji has 110 samples.
Class african_hunting_dog has 86 samples.
Class boxer has 75 samples.
Class flat-coated_retriever has 72 samples.
Class bedlington_terrier has 89 samples.
Class clumber has 80 samples.
Class shih-tzu has 112 samples.
Class curly-coated_retriever has 72 samples.
Class wire-haired_fox_terrier has 82 samples.
Class briard has 66 samples.
Class dingo has 80 samples.
Class malinois has 73 samples.
Class sussex_spaniel has 78 samples.
Class bull_mastiff has 75 samples.
Class tibetan_terrier has 107 samples.
Class border_collie has 72 samples.
Class bouvier_des_flandres has 86 samples.
Class maltese_dog has 117 samples.
Class appenzeller has 78 samples.
```

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

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

```
[14] datagen = ImageDataGenerator(rescale=1./255,
         shear_range=0.2,
         zoom_range=0.2,
         horizontal_flip=True,
         validation_split=0.2) # set validation split


     train_generator = datagen.flow_from_directory(
         directory=train_dir,
         target_size=(331, 331),
         batch_size=32,
         class_mode='categorical',
         subset='training')

     validation_generator = datagen.flow_from_directory(
         directory=train_dir,
         target_size=(331, 331),
         batch_size=32,
         class_mode='categorical',
         subset='validation')
```

```
Found 8221 images belonging to 120 classes.
Found 2001 images belonging to 120 classes.
```

Milestone 3: Model Building

Now it's time to build our Convolutional Neural Networking using inceptionv3 which contains an input layer along with the convolution, max-pooling, and finally an output layer.

Activity 1: Importing the Model Building Libraries

Importing the necessary libraries

```
from tensorflow.keras.applications import NASNetLarge
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

Activity 2: Initializing the model:

Load the pre-trained NASnetLarge model

```
# Load the pre-trained NASNetLarge model
nasnet_large_model = NASNetLarge(weights='imagenet', include_top=False, input_shape=(331, 331, 3))

# Set the base model to non-trainable
nasnet_large_model.trainable = False
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/nasnet/NASNet-large-no-top.h5
343610240/343610240 [==============================] - 17s 0us/step
```

Activity 3: Adding Fully connected Layers

● As the input image contains three channels, we are specifying the input shape as (331,331,3).

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

```
[17] # Build your custom model on top
     model = Sequential()
     model.add(nasnet_large_model)
     model.add(GlobalAveragePooling2D())
     model.add(Dropout(0.5))  # Increase dropout rate for more regularization
     model.add(Dense(120, activation='softmax', kernel_regularizer=l2(0.001)))  # Apply L2 regularization
```

dropout layer stops overfitting the model.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.

Activity 4: 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.

```
# Compile the model with a lower learning rate
model.compile(optimizer=Adam(learning_rate=0.0001), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Apply early stopping and learning rate reduction
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=1e-6)
```

```
[34] model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 NASNet (Functional)         (None, 11, 11, 4032)      84916818

 global_average_pooling2d (  (None, 4032)              0
 GlobalAveragePooling2D)

 dropout (Dropout)           (None, 4032)              0

 dense (Dense)               (None, 120)               483960

=================================================================
Total params: 85400778 (325.78 MB)
Trainable params: 483960 (1.85 MB)
Non-trainable params: 84916818 (323.93 MB)
_________________________________________________________________
```

Activity 5: 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 40 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.

```
batch_size=32
# Train the model
history = model.fit(train_generator,
                    steps_per_epoch=train_generator.samples // batch_size,
                    validation_data=validation_generator,
                    validation_steps=validation_generator.samples // batch_size,
                    epochs=40,
                    callbacks=[early_stop, reduce_lr],  # Add early stopping and learning rate reduction callbacks
                    verbose=1)
```

```
Epoch 1/40
256/256 [==============================] - 449s 2s/step - loss: 3.6263 - accuracy: 0.5901 - val_loss: 2.2935 - val_accuracy: 0.9128 - lr: 1.0000e-04
Epoch 2/40
256/256 [==============================] - 394s 2s/step - loss: 1.6299 - accuracy: 0.9171 - val_loss: 1.1474 - val_accuracy: 0.9259 - lr: 1.0000e-04
Epoch 3/40
256/256 [==============================] - 378s 1s/step - loss: 0.9912 - accuracy: 0.9262 - val_loss: 0.8556 - val_accuracy: 0.9284 - lr: 1.0000e-04
Epoch 4/40
256/256 [==============================] - 379s 1s/step - loss: 0.8025 - accuracy: 0.9345 - val_loss: 0.7513 - val_accuracy: 0.9304 - lr: 1.0000e-04
Epoch 5/40
256/256 [==============================] - 378s 1s/step - loss: 0.7223 - accuracy: 0.9342 - val_loss: 0.6937 - val_accuracy: 0.9345 - lr: 1.0000e-04
Epoch 6/40
256/256 [==============================] - 379s 1s/step - loss: 0.6718 - accuracy: 0.9350 - val_loss: 0.6613 - val_accuracy: 0.9269 - lr: 1.0000e-04
Epoch 7/40
256/256 [==============================] - 387s 2s/step - loss: 0.6343 - accuracy: 0.9385 - val_loss: 0.6308 - val_accuracy: 0.9350 - lr: 1.0000e-04
Epoch 8/40
256/256 [==============================] - 378s 1s/step - loss: 0.6036 - accuracy: 0.9374 - val_loss: 0.6009 - val_accuracy: 0.9340 - lr: 1.0000e-04
Epoch 9/40
256/256 [==============================] - 381s 1s/step - loss: 0.5775 - accuracy: 0.9389 - val_loss: 0.5805 - val_accuracy: 0.9365 - lr: 1.0000e-04
Epoch 10/40
256/256 [==============================] - 383s 1s/step - loss: 0.5547 - accuracy: 0.9402 - val_loss: 0.5574 - val_accuracy: 0.9380 - lr: 1.0000e-04
Epoch 11/40
256/256 [==============================] - 408s 2s/step - loss: 0.5343 - accuracy: 0.9392 - val_loss: 0.5351 - val_accuracy: 0.9410 - lr: 1.0000e-04
Epoch 12/40
256/256 [==============================] - 409s 2s/step - loss: 0.5152 - accuracy: 0.9421 - val_loss: 0.5267 - val_accuracy: 0.9350 - lr: 1.0000e-04
Epoch 13/40
256/256 [==============================] - 406s 2s/step - loss: 0.5012 - accuracy: 0.9420 - val_loss: 0.5181 - val_accuracy: 0.9345 - lr: 1.0000e-04
Epoch 14/40
256/256 [==============================] - 396s 2s/step - loss: 0.4857 - accuracy: 0.9435 - val_loss: 0.5062 - val_accuracy: 0.9335 - lr: 1.0000e-04
Epoch 15/40
256/256 [==============================] - 398s 2s/step - loss: 0.4708 - accuracy: 0.9441 - val_loss: 0.4916 - val_accuracy: 0.9370 - lr: 1.0000e-04
Epoch 16/40
256/256 [==============================] - 397s 2s/step - loss: 0.4583 - accuracy: 0.9444 - val_loss: 0.4834 - val_accuracy: 0.9380 - lr: 1.0000e-04
Epoch 17/40
256/256 [==============================] - 398s 2s/step - loss: 0.4471 - accuracy: 0.9448 - val_loss: 0.4731 - val_accuracy: 0.9345 - lr: 1.0000e-04
Epoch 18/40
256/256 [==============================] - 398s 2s/step - loss: 0.4330 - accuracy: 0.9482 - val_loss: 0.4723 - val_accuracy: 0.9365 - lr: 1.0000e-04
Epoch 19/40
256/256 [==============================] - 407s 2s/step - loss: 0.4249 - accuracy: 0.9496 - val_loss: 0.4573 - val_accuracy: 0.9370 - lr: 1.0000e-04
Epoch 20/40
256/256 [==============================] - 397s 2s/step - loss: 0.4222 - accuracy: 0.9461 - val_loss: 0.4506 - val_accuracy: 0.9360 - lr: 1.0000e-04
Epoch 21/40
256/256 [==============================] - 401s 2s/step - loss: 0.4139 - accuracy: 0.9466 - val_loss: 0.4579 - val_accuracy: 0.9289 - lr: 1.0000e-04
Epoch 22/40
256/256 [==============================] - 410s 2s/step - loss: 0.4045 - accuracy: 0.9470 - val_loss: 0.4341 - val_accuracy: 0.9410 - lr: 1.0000e-04
Epoch 23/40
256/256 [==============================] - 410s 2s/step - loss: 0.3968 - accuracy: 0.9461 - val_loss: 0.4370 - val_accuracy: 0.9360 - lr: 1.0000e-04
Epoch 24/40
256/256 [==============================] - 409s 2s/step - loss: 0.3912 - accuracy: 0.9497 - val_loss: 0.4292 - val_accuracy: 0.9340 - lr: 1.0000e-04
Epoch 25/40
256/256 [==============================] - 408s 2s/step - loss: 0.3869 - accuracy: 0.9509 - val_loss: 0.4328 - val_accuracy: 0.9355 - lr: 1.0000e-04
Epoch 26/40
256/256 [==============================] - 414s 2s/step - loss: 0.3831 - accuracy: 0.9501 - val_loss: 0.4296 - val_accuracy: 0.9370 - lr: 1.0000e-04
Epoch 27/40
256/256 [==============================] - 400s 2s/step - loss: 0.3740 - accuracy: 0.9510 - val_loss: 0.4165 - val_accuracy: 0.9370 - lr: 2.0000e-05
Epoch 28/40
256/256 [==============================] - 468s 2s/step - loss: 0.3767 - accuracy: 0.9483 - val_loss: 0.4234 - val_accuracy: 0.9370 - lr: 2.0000e-05
Epoch 29/40
256/256 [==============================] - 420s 2s/step - loss: 0.3726 - accuracy: 0.9490 - val_loss: 0.4195 - val_accuracy: 0.9370 - lr: 2.0000e-05
Epoch 30/40
256/256 [==============================] - 400s 2s/step - loss: 0.3688 - accuracy: 0.9515 - val_loss: 0.4261 - val_accuracy: 0.9335 - lr: 4.0000e-06
```

Activity 6: 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.

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.

```
model.save("NASNetLarge.h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This f
  saving_api.save_model(
```

Activity 7: 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.

```
[20] import os

     # Define the path to the 'subset/train' folder
     train_folder_path = '/content/subset/train'

     # Get a list of class names (subfolder names)
     class_names = sorted(os.listdir(train_folder_path))

     # Print the class names
     print(class_names)
```

```
['affenpinscher', 'afghan_hound', 'african_hunting_dog', 'airedale', 'american_staffordshire_terrier', 'appenzeller', 'australian_terrier', 'basenji', 'basse
```

```
[30] import tensorflow as tf
     from tensorflow.keras.preprocessing import image
     import numpy as np

     # Load and preprocess the image
     img = image.load_img("/content/subset/train/affenpinscher/00ca18751837cd6a22813f8e221f7819.jpg", target_size=(331, 331))
     x = image.img_to_array(img)
     x = np.expand_dims(x, axis=0)
     x = tf.keras.applications.nasnet.preprocess_input(x)

     # Make predictions
     predictions = model.predict(x)

     # Decode the predictions to get the class label
     predicted_class_index = np.argmax(predictions)
     predicted_class_name = class_names[predicted_class_index]

     print(f"Predicted class: {predicted_class_name}")
```

```
1/1 [==============================] - 0s 57ms/step
Predicted class: affenpinscher
```

```
[32]

     # Load and preprocess the image
     img = image.load_img("/content/subset/train/american_staffordshire_terrier/041d0d6a8d110b35a3795dd5c68f9a36.jpg", target_size=(331, 331))
     x = image.img_to_array(img)
     x = np.expand_dims(x, axis=0)
     x = tf.keras.applications.nasnet.preprocess_input(x)

     predictions = model.predict(x)
     predicted_class_index = np.argmax(predictions)
     predicted_class_name = class_names[predicted_class_index]

     print(f"Predicted class: {predicted_class_name}")
```

```
1/1 [==============================] - 0s 55ms/step
Predicted class: american_staffordshire_terrier
```

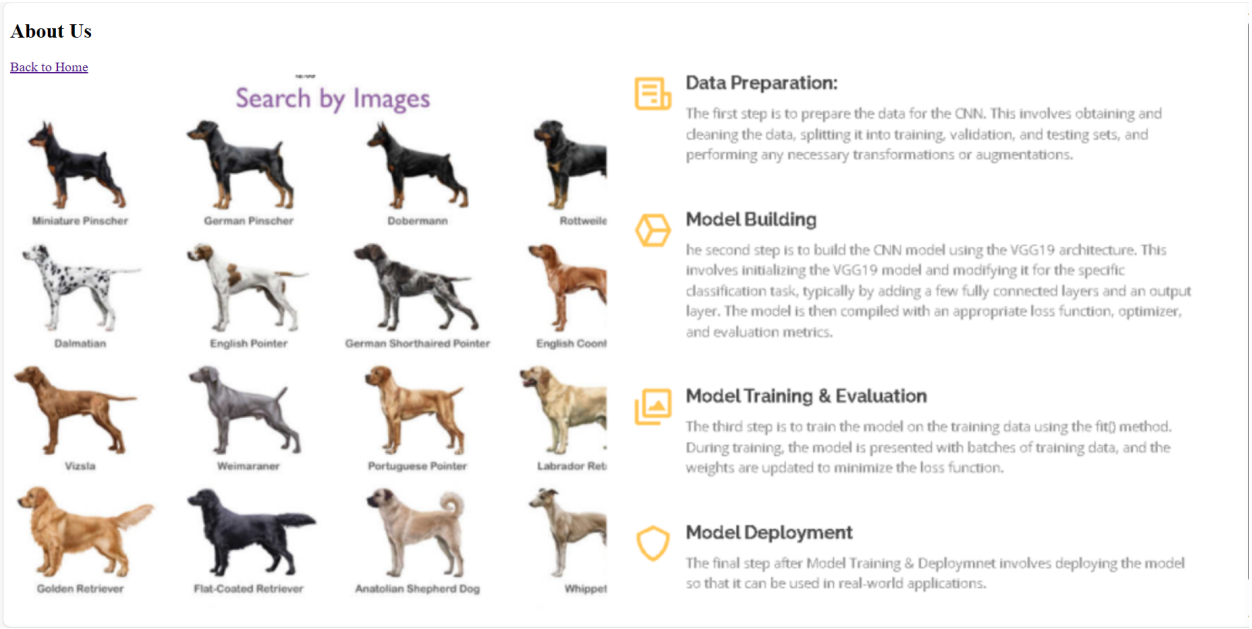
Milestone 4: Application Building

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

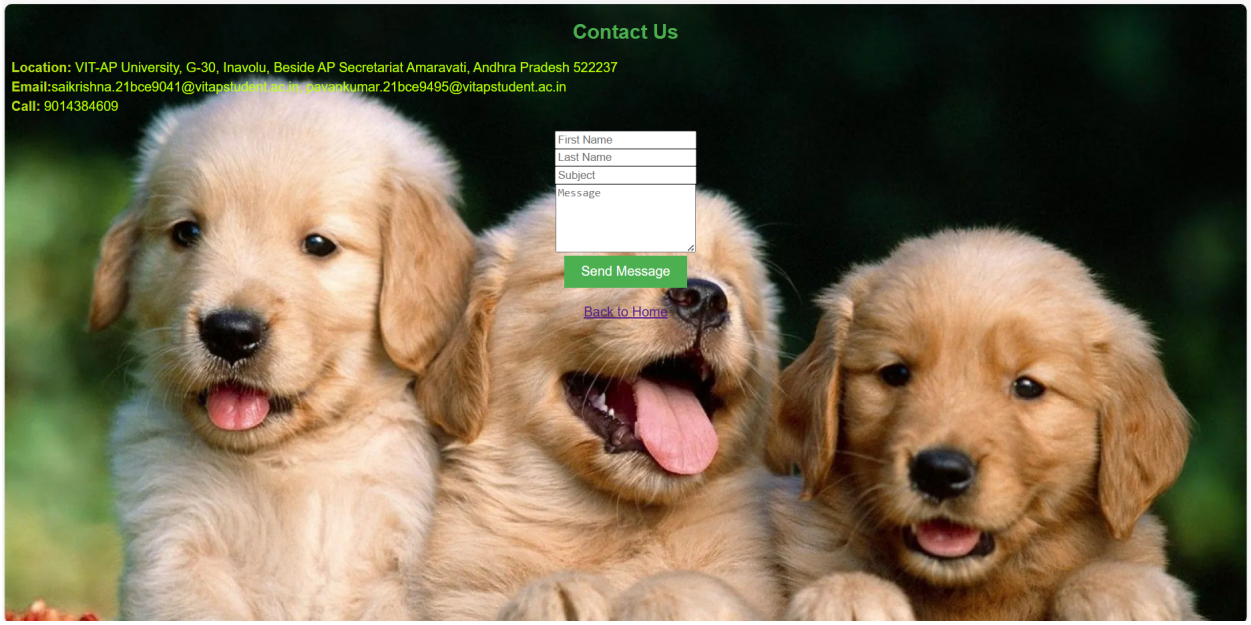
HTML page to notify the user. Whenever the user interacts with the UI and selects the “Inspect” button, the next page is opened where the user chooses the image and predicts the output.

index.html looks like this

About Section:



Contact:



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.

```
app.py > ...
1   from flask import Flask, render_template, request
2   from tensorflow.keras.preprocessing import image
3   import numpy as np
4   import tensorflow as tf
5
```

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

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.

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

```python
@app.route('/search', methods=['POST'])
def search():
    file = request.files['file']
    file.save('uploaded_image.jpg')
```

```python
img = image.load_img('uploaded_image.jpg', target_size=(331, 331))  # Load the uploaded image
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = tf.keras.applications.nasnet.preprocess_input(x)
predictions = model.predict(x)
predicted_class_index = np.argmax(predictions)
breed = class_names[predicted_class_index]
```

Showcasing prediction on UI:

```python
    return render_template('prediction.html', breed=breed)
```

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 119.) 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.

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

Activity 3:Run the application:

open vs code

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.

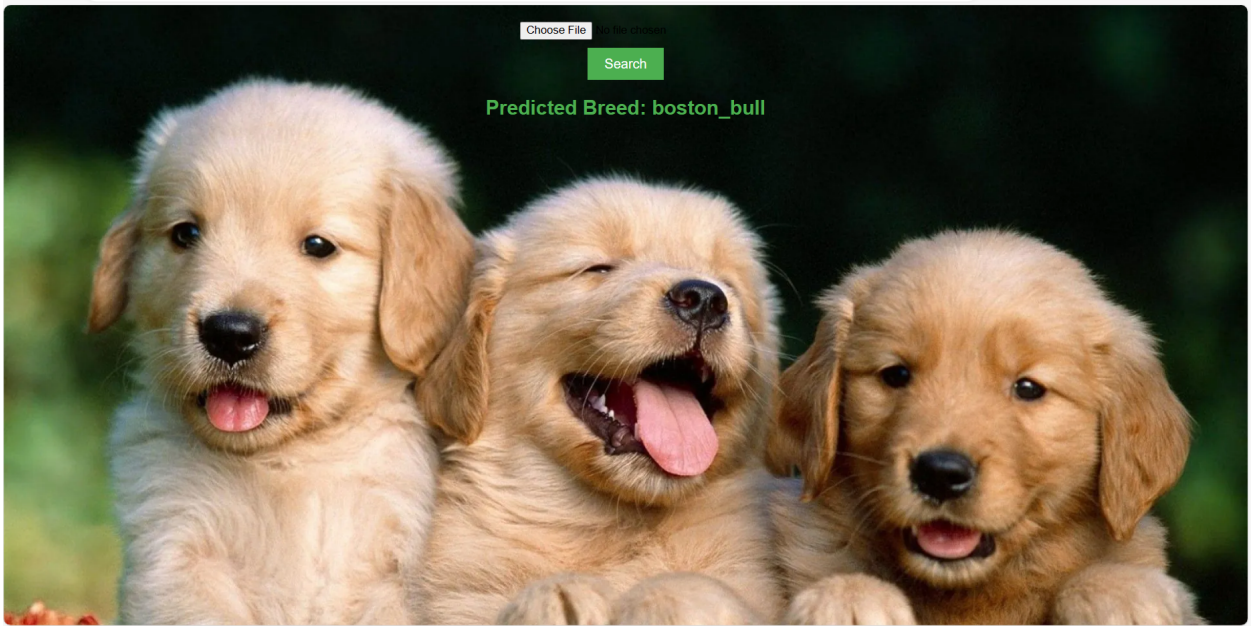
● Enter the values, click on the predict button and see the result/prediction on the web page.

Then it will run on localhost: 5000

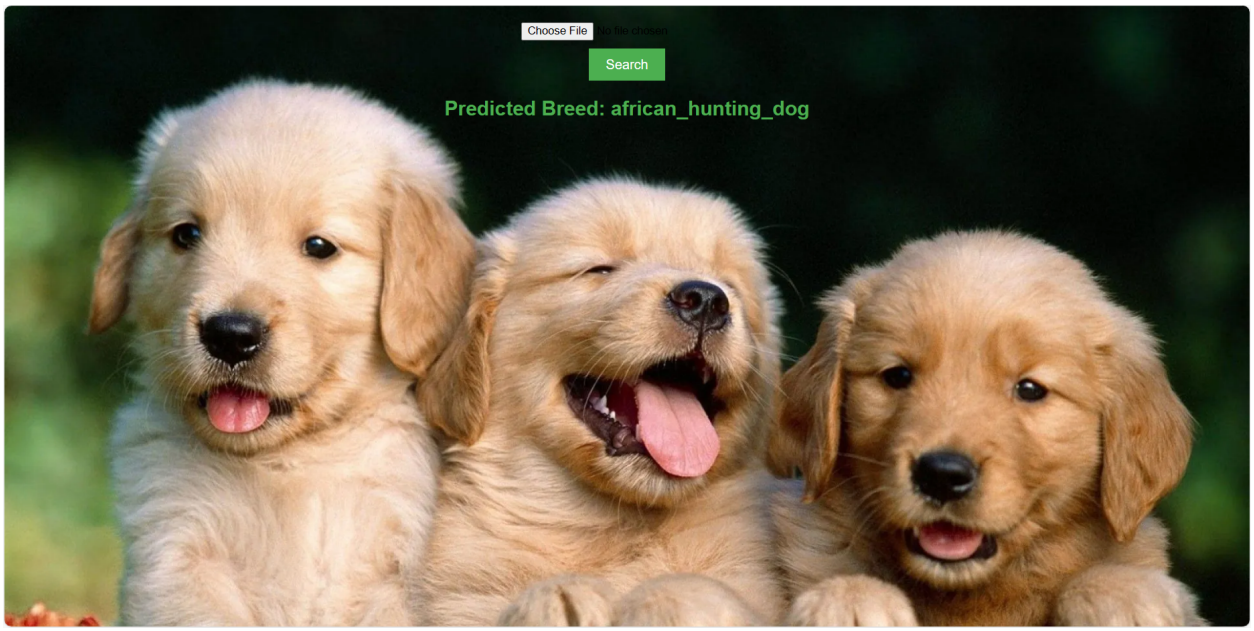
```
PS C:\Users\gogul\Downloads\Dog Breed> & C:/Users/gogul/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/gogul/Downloads/Dog Breed/app.py"
2023-11-03 20:42:28.523407: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical op
erations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
2023-11-03 20:42:43.806111: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instructions in performance-critical op
erations.
To enable the following instructions: SSE SSE2 SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

FINAL OUTPUT:

OUTPUT-1:



OUTPUT-2:


Choose File
No file chosen
Search
Predicted Breed: african_hunting_dog