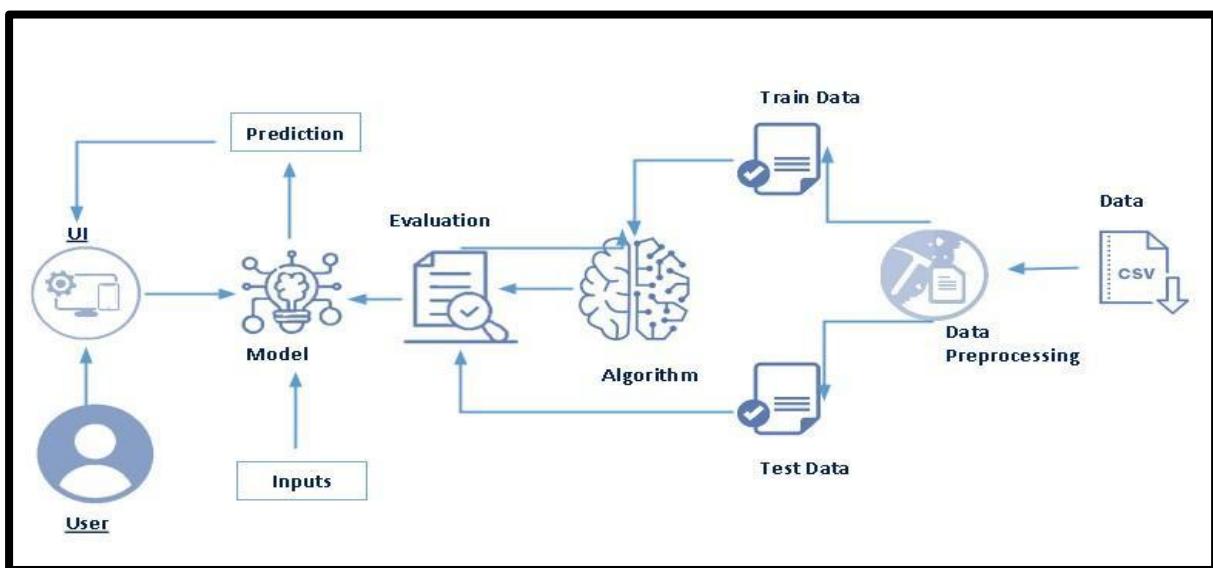


# Dog Breed Identification using Transfer Learning

## Project Description:

In recent years, computer vision and deep learning have greatly improved image recognition tasks. Identifying dog breeds manually is challenging due to similar visual features, subtle texture differences, and variations in pose. Dog breed classification plays an important role in veterinary care, animal rescue and adoption platforms, pet-care recommendation systems, and wildlife research. Traditional image-processing methods are often time-consuming and less accurate. To address this, the proposed system uses deep learning with transfer learning based on MobileNetV2 to automatically classify dog breeds from images. The model applies CNN-based feature extraction to learn fine-grained visual patterns and deliver fast, accurate predictions through a web-based interface, enabling an accessible and intelligent solution for automated animal recognition.

## Technical Architecture:



## Pre requisites:

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 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:**  
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 tensorflow==2.12.0” and click enter.  
Type “pip install keras==2.12.0” and click enter.  
Type “pip install Flask” and click enter.

## 2. Deep Learning Concepts

- **CNN:** A convolutional neural network is a class of deep neural networks, most commonly applied to analysing visual imagery. [CNN Basic](#)
- **MobileNetV2:** MobileNetV2 is a lightweight deep convolutional neural network architecture designed for efficient image classification.  
It uses depth wise separable convolutions and inverted residual blocks to reduce computation while maintaining high accuracy.  
In this project, MobileNetV2 is applied through transfer learning to accurately classify different dog breeds from images with faster training and lower resource usage.
- **Flask:** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.  
[Flask Basics](#)

## 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 upload or choose a dog image.
- The selected image is processed and analyzed by the deep learning model integrated with the Flask application.
- A **CNN-based MobileNetV2 transfer learning model** analyzes the image, and the predicted dog breed is displayed on the Flask UI. To accomplish this, the following activities and tasks are performed:

### Data Collection

- Collect diverse dog breed images from **public datasets such as Kaggle**.
- Ensure sufficient images for each breed class.

### Dataset Preparation

- Organize images into **Train and Test folders**.
- Maintain proper class-wise directory structure.

### Data Preprocessing

- Import the **ImageDataGenerator** library.
- Configure **ImageDataGenerator** for:
  - Image resizing
  - Normalization
  - Data augmentation
- Apply ImageDataGenerator to **training and testing datasets**.

### Model Building

- Import required **TensorFlow / Keras libraries**.
- Load **MobileNetV2 pre-trained model (transfer learning)**.
- Freeze base layers and initialize the model.
- Add **custom fully connected classification layers**.

- Configure the learning process (optimizer, loss, metrics).

## Model Training and Testing

- Train the model using the **training dataset**.
  - Validate performance using the **test dataset**.
  - Evaluate accuracy and prediction performance.

## Model Saving

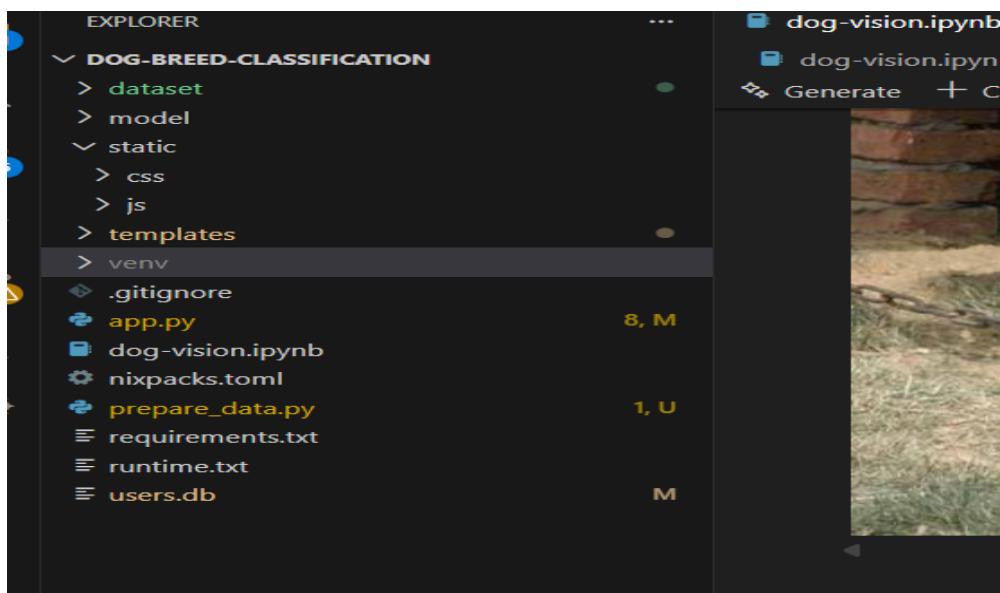
- Save the trained model in **.h5 format** for deployment and inference.

## **Application Development**

- Create **HTML templates** for image upload and prediction display.
  - Develop **Flask backend code** to:
    - Load the trained model
    - Accept uploaded images
    - Perform prediction
    - Display predicted dog breed on the UI.

## Project Structure:

Create the Project folder which contains files as shown below



- We are building a Flask-based web application that uses HTML pages stored inside the templates folder for the user interface and a Python script (app.py) to handle routing, prediction logic, and backend processing.
- The trained deep learning model (MobileNetV2) is saved in .h5 format inside the model folder, and this saved model is loaded in the Flask application to perform dog breed prediction during runtime.
- The project includes dataset preparation, model training scripts, and deployment configuration files:
  - prepare\_data.py for organizing dataset images
  - dog-vision.ipynb / training code for model training using transfer learning
  - Deployment files (requirements, runtime, etc.)

## Milestone 1: Data Collection & Image Preprocessing

Machine Learning and Deep Learning models strongly depend on **high-quality data**, which is the most important factor for accurate model training and prediction. This phase focuses on **collecting the required dog image dataset** used for training the classification model.

### Activity 1: Download the Dataset

There are many well-known open-source platforms for collecting datasets, such as:

- **Kaggle**
- **UCI Machine Learning Repository**
- Other public research datasets

For this project, we use the **Dog Breed Identification dataset** obtained from **Kaggle**, which contains thousands of dog images belonging to multiple breeds. This dataset is used to train a **deep learning model with transfer learning (MobileNetV2)** for automatic dog breed prediction.

### DatasetLink:

<https://www.kaggle.com/competitions/dog-breed-identification/data?select=train>

After downloading the dataset, the next step is to **understand the structure, distribution, and characteristics of the dog breed images** using visualization and analysis techniques. Proper data understanding helps in selecting the right preprocessing methods and improves model performance.

**Note:** Many visualization techniques exist. In this project, only essential and meaningful techniques are used for analysis.

### Activity : Importing the Libraries

The required Python libraries are imported for **data handling, visualization, and image processing**, such as:

- **NumPy** – numerical operations
- **Pandas** – reading labels and dataset information
- **TensorFlow / Keras** – deep learning and image processing

```
import os
from flask import Flask, request, jsonify, render_template, redirect, url_for, session, flash
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow_hub as hub
from PIL import Image
import io
import base64
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
```

### Activity : Reading the Dataset

The dog breed dataset contains:

- **Images of dogs**
- **labels.csv file** mapping image IDs to breed names

The dataset is read using **Pandas read\_csv()** for labels and **directory-based image loading** for training.

This helps in:

- Understanding **number of images per breed**
- Checking **missing or imbalanced classes**
- Preparing data for preprocessing

...		<b>id</b>	<b>breed</b>
count		10222	10222
unique		10222	120
top	ffff43b07992508bc822f33d8ffd902ae	scottish_deerhound	
freq		1	126
		<b>id</b>	<b>breed</b>
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull	
1	001513dfcb2ffafc82cccf4d8bbaba97	dingo	
2	001cdf01b096e06d78e9e5112d419397	pekinese	
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick	
4	0021f9ceb3235effd7fcde7f7538ed62	golden_retriever	

labels_csv.head()			
...		<b>id</b>	<b>breed</b>
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull	
1	001513dfcb2ffafc82cccf4d8bbaba97	dingo	
2	001cdf01b096e06d78e9e5112d419397	pekinese	
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick	
4	0021f9ceb3235effd7fcde7f7538ed62	golden_retriever	

## Activity : Descriptive Analysis

Descriptive statistics summarize the dataset using:

- **Total number of images**
- **Number of dog breeds (classes)**
- **Minimum, maximum, and average images per class**

Using **Pandas describe()** and dataset inspection, we confirm:

- Dataset is suitable for **deep learning classification**
- Requires **normalization and augmentation** before training

## Activity : Organizing the Images into Different Classes

In this step, the Dog Breed Identification dataset collected from Kaggle is prepared for training the deep learning model.

Each dog image is mapped to its corresponding breed label using the labels.csv file provided in the dataset.

Based on these labels, the images are arranged into separate folders for each dog breed, creating a structured directory format required for CNN training.

This organization enables the ImageDataGenerator and MobileNetV2 transfer learning model to correctly learn visual patterns for multiclass dog breed classification.



## Activity : Import Image Preprocessing Utilities

In this project, instead of using **ImageDataGenerator**, the system directly imports **TensorFlow/Keras image preprocessing utilities** to prepare images for prediction using the **MobileNetV2 transfer learning model**.

These utilities perform:

- Image loading from the uploaded file
- Resizing to **224 × 224** pixels (MobileNetV2 input size)
- Conversion to array format
- Normalization using **MobileNetV2 preprocessing function**

## Activity : Configure Image Preprocessing Pipeline

In this project, instead of configuring an ImageDataGenerator, the preprocessing pipeline is designed specifically for MobileNetV2 transfer learning inference.

The configuration includes:

- Loading the uploaded dog image using PIL / TensorFlow utilities
- Resizing the image to  $224 \times 224$  pixels, which is the required input size for MobileNetV2

- Converting the image into a numerical array suitable for neural network processing
- Applying MobileNetV2 preprocessing normalization, which scales pixel values and prepares the image in the same format used during model training

This preprocessing pipeline ensures:

- Compatibility with the pre-trained MobileNetV2 model
- Fast real-time prediction inside the Flask web application
- Consistent and reliable classification performance

Thus, the project replaces traditional `ImageDataGenerator` configuration with a direct MobileNetV2-specific preprocessing workflow optimized for deployment and inference.

### **Activity : Splitting data into train and test**

In this project, traditional **ImageDataGenerator-based dataset loading is not used**. Instead, the dataset is handled using **direct file paths and labels**, followed by splitting with **scikit-learn**.

#### **Process followed**

1. **Image paths and corresponding breed labels** are stored in arrays.
2. The dataset is divided into **training and validation sets** using:
  - `train_test_split()` from `sklearn.model_selection`
  - **80% data for training**
  - **20% data for validation**
  - Random state is fixed for **reproducibility**.
3. The resulting split produces:
  - **X\_train, y\_train → Training images and labels**
  - **X\_val, y\_val → Validation images and labels**

This approach provides:

- **Simple and memory-efficient dataset handling**
- **Better control over training and validation distribution**
- **Compatibility with MobileNetV2 preprocessing pipeline**

Thus, instead of applying **ImageDataGenerator functionality**, the project relies on **manual dataset splitting and preprocessing**, which is sufficient for **transfer learning-based dog breed classification**

```
# Let's split our data into train and validation sets
from sklearn.model_selection import train_test_split

# Split them into training and validation of total size NUM_IMAGES
X_train, X_val, y_train, y_val = train_test_split(X[:NUM_IMAGES], y[:NUM_IMAGES], test_size=0.2, random_state=42)

len(X_train), len(y_train), len(X_val), len(y_val)
...
... (800, 800, 200, 200)

# Let's have a look at the training data
X_train[:5], y_train[:2]
...
(['drive/My Drive/Dog Vision/train/00bee065dcec471f26394855c5c2f3de.jpg',
 'drive/My Drive/Dog Vision/train/0d2f9e12a2611d91a339074c8154.jpg',
 'drive/My Drive/Dog Vision/train/1108e48ce3e2d7d7fb527ae6e40ab486.jpg',
 'drive/My Drive/Dog Vision/train/1108e48ce3e2d7d7fb527ae6e40ab486.jpg',
 'drive/My Drive/Dog Vision/train/1108e48ce3e2d7d7fb527ae6e40ab486.jpg'])
```

## Milestone 2: Model Building

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. for this project we are applying four classification algorithms. The best model is saved based on its performance.

### Activity 1: Importing the Model Building Libraries

To construct the deep learning model, the required libraries are imported, including:

- TensorFlow / Keras for neural network construction
- NumPy for numerical operations
- MobileNetV2 from tensorflow.keras.applications for transfer learning
- Image preprocessing utilities for resizing and normalization

These libraries enable efficient image-based classification using CNN architecture.

### Activity 2: Importing the MobileNetV2 Pre-trained Model

Instead of VGG19, this project uses MobileNetV2, a lightweight and efficient convolutional neural network designed for image classification.

- Pre-trained on ImageNet dataset
- Provides high accuracy with low computational cost
- Suitable for real-time web deployment

Only the feature extraction layers are reused, enabling transfer learning.

### Activity 3: Initializing the Transfer Learning Model

The MobileNetV2 base model is loaded with:

- Pre-trained ImageNet weights
- Top layers removed (include\_top=False)

- Input image size configured to  $224 \times 224$

This allows the network to learn dog-breed-specific features while retaining powerful generic visual features.

```

import tensorflow as tf
import tensorflow_hub as hub

# Create a function which builds a Keras model
def create_model(input_shape=INPUT_SHAPE, output_shape=OUTPUT_SHAPE, model_url=MODEL_URL):
    print("Building model with:", model_url)

    # Setup the model layers
    model = tf.keras.Sequential([
        tf.keras.layers.Lambda(hub_layer_fn, name="hub_lambda"), # Wrap the hub.KerasLayer in Lambda
        tf.keras.layers.Dense(units=output_shape, activation="softmax") # Output layer with softmax activation
    ])

    # Compile the model
    model.compile(
        loss=tf.keras.losses.CategoricalCrossentropy(),
        optimizer=tf.keras.optimizers.Adam(),
        metrics=["accuracy"]
    )

    # Build the model
    model.build((None, *input_shape)) # Ensure input shape is unpacked correctly

    return model

```

#### Activity 4: Adding Classification Layer & Model Summary

After loading the **MobileNetV2** model, a **Dense softmax classification layer** is added to match the **120 dog breed classes**.

This connects the feature extractor to the final prediction output.

The **model summary** is then used to verify:

- Correct layer structure and output shapes
- Total and trainable parameters
- Readiness of the model for training

```

...
Model: "sequential_16"

...


| Layer (type)        | Output Shape | Param # |
|---------------------|--------------|---------|
| hub_lambda (Lambda) | (None, 1001) | 0       |
| dense_16 (Dense)    | (None, 120)  | 120,240 |


...
Total params: 120,240 (469.69 KB)
...
Trainable params: 120,240 (469.69 KB)
...
Non-trainable params: 0 (0.00 B)

```

## Activity 5: Configuring the Learning Process

The model training configuration includes:

- Loss function: Categorical Cross-Entropy
- Optimizer: Adam optimizer
- Evaluation metric: Accuracy

This setup ensures stable convergence and accurate multi-class classification.

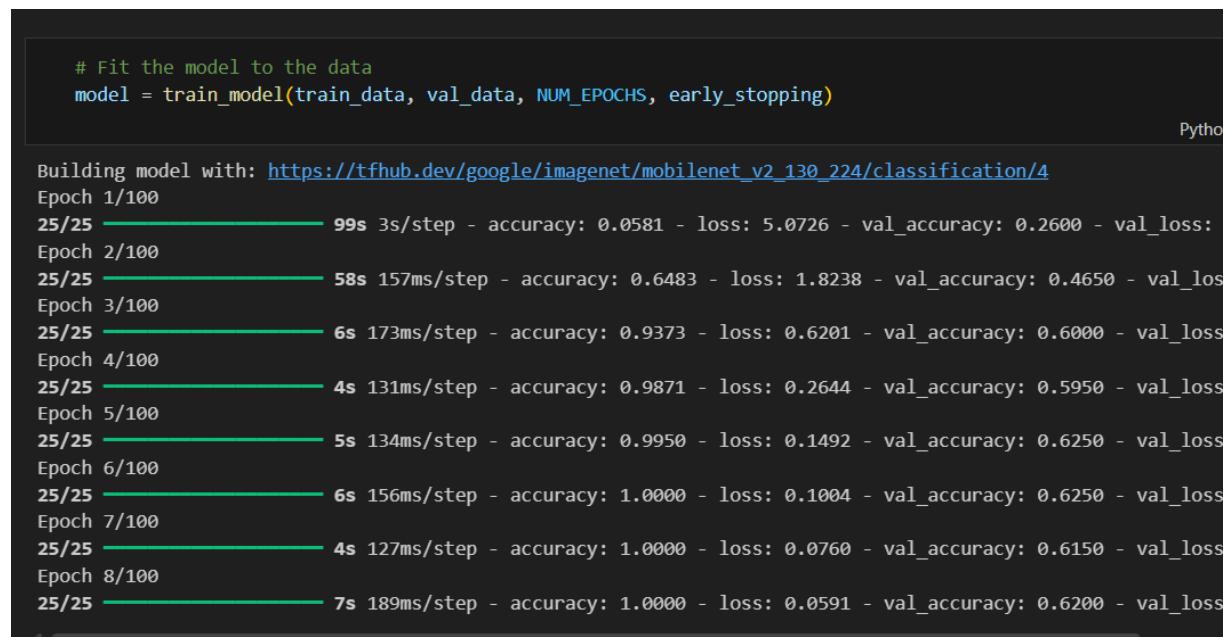
## Activity 6: Training the Model

The MobileNetV2-based deep learning model is trained using the prepared training and validation datasets.

During training, the system learns breed-specific visual patterns from images over multiple epochs.

Accuracy and loss values are monitored to evaluate performance and ensure proper learning.

Early stopping is applied to prevent overfitting and improve generalization.



```
# Fit the model to the data
model = train_model(train_data, val_data, NUM_EPOCHS, early_stopping)
```

Building model with: [https://tfhub.dev/google/imagenet/mobilenet\\_v2\\_130\\_224/classification/4](https://tfhub.dev/google/imagenet/mobilenet_v2_130_224/classification/4)

Epoch 1/100  
25/25 99s 3s/step - accuracy: 0.0581 - loss: 5.0726 - val\_accuracy: 0.2600 - val\_loss:  
Epoch 2/100  
25/25 58s 157ms/step - accuracy: 0.6483 - loss: 1.8238 - val\_accuracy: 0.4650 - val\_loss:  
Epoch 3/100  
25/25 6s 173ms/step - accuracy: 0.9373 - loss: 0.6201 - val\_accuracy: 0.6000 - val\_loss:  
Epoch 4/100  
25/25 4s 131ms/step - accuracy: 0.9871 - loss: 0.2644 - val\_accuracy: 0.5950 - val\_loss:  
Epoch 5/100  
25/25 5s 134ms/step - accuracy: 0.9950 - loss: 0.1492 - val\_accuracy: 0.6250 - val\_loss:  
Epoch 6/100  
25/25 6s 156ms/step - accuracy: 1.0000 - loss: 0.1004 - val\_accuracy: 0.6250 - val\_loss:  
Epoch 7/100  
25/25 4s 127ms/step - accuracy: 1.0000 - loss: 0.0760 - val\_accuracy: 0.6150 - val\_loss:  
Epoch 8/100  
25/25 7s 189ms/step - accuracy: 1.0000 - loss: 0.0591 - val\_accuracy: 0.6200 - val\_loss:

## Activity 7: Saving the Trained Model

After successful training:

- The trained MobileNetV2-based model is saved.
- This saved model is later **loaded inside the Flask application**
- Enables **real-time prediction without retraining**

## Activity 8: Testing and Prediction

The saved model is evaluated using:

- Validation images not seen during training
- Prediction probabilities generated via Softmax output
- Final dog breed label displayed in the web interface

This confirms the model's generalization capability and real-time usability.

A screenshot of a Jupyter Notebook interface. At the top, there are tabs for 'Generate', 'Code', 'Markdown', 'Run All', 'Clear All Outputs', and 'Outline'. A dropdown menu 'Select Kernel' is visible. Below the tabs, the status bar shows '324/324' and '2238s 7s/step'. The main area contains three code cells. The first cell shows the output of `test_predictions[:10]`, which is a 2D array of floating-point numbers. The second cell shows the output of `get_pred_label(test_predictions[0])`, which is the string '`'soft-coated_wheaten_terrier'`'. The third cell shows the output of `test_predictions[0].max()`. The entire screenshot is framed by a dark border.

```
... 324/324 2238s 7s/step
... Select Kernel
... test_predictions[:10]
Python
...
array([[2.1132093e-11, 3.7077545e-09, 7.1800510e-11, ... , 2.2235262e-10,
       1.6396466e-11, 8.2923252e-11],
      [2.6324756e-10, 1.6052845e-11, 1.4962432e-12, ... , 4.1345946e-12,
       8.4858223e-12, 5.9450218e-12],
      [3.3900855e-09, 1.3403331e-10, 4.4861707e-08, ... , 1.4973245e-06,
       1.9893285e-05, 6.8071948e-08],
      ... ,
      [2.1980105e-08, 9.2089767e-09, 2.4912410e-09, ... , 3.9026932e-07,
       1.1340238e-10, 5.4193650e-11],
      [1.2264234e-12, 9.9979109e-01, 1.5065426e-08, ... , 6.5629444e-09,
       4.8026649e-11, 7.5082602e-07],
      [2.2741311e-08, 1.9328403e-05, 1.9725517e-10, ... , 8.8675783e-08,
       6.7815363e-12, 2.4887495e-10]], dtype=float32)

... get_pred_label(test_predictions[0])
Python
...
'soft-coated_wheaten_terrier'

... test_predictions[0].max()
```

## Milestone 3: Application Building

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where We upload an image for predicting the breed of the dog. The Image is given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building serverside script

### Activity1: Create HTML Pages

Design simple and user-friendly web pages using HTML and CSS. These pages allow users to upload a dog image and view the predicted breed result clearly.

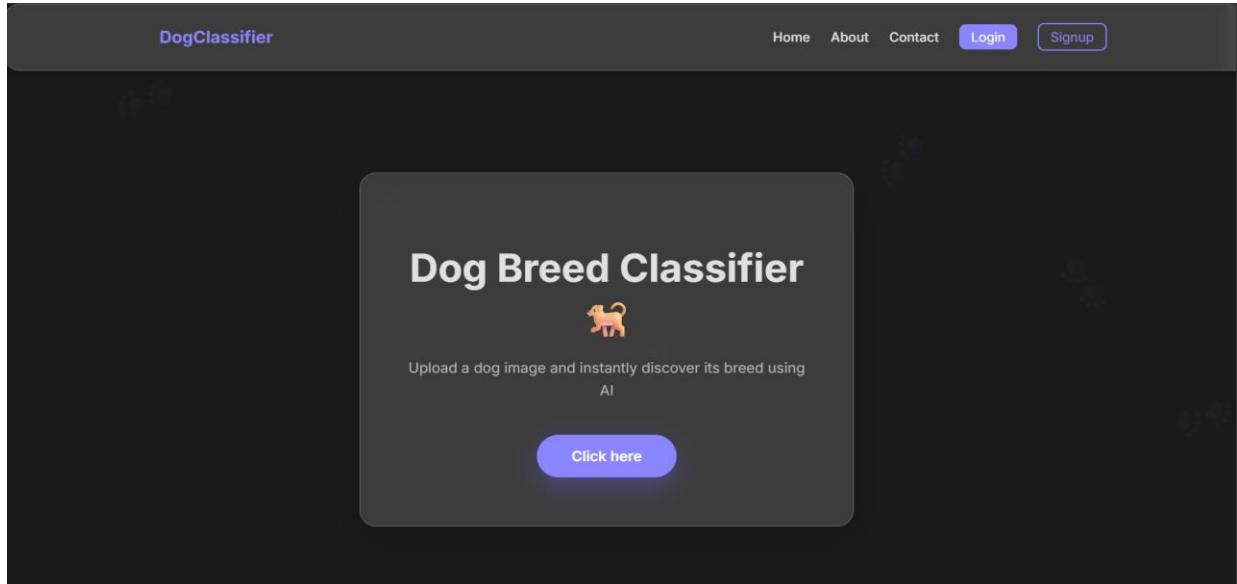
For this project create three HTML files namely

- home.html
- contact.html
- about.html

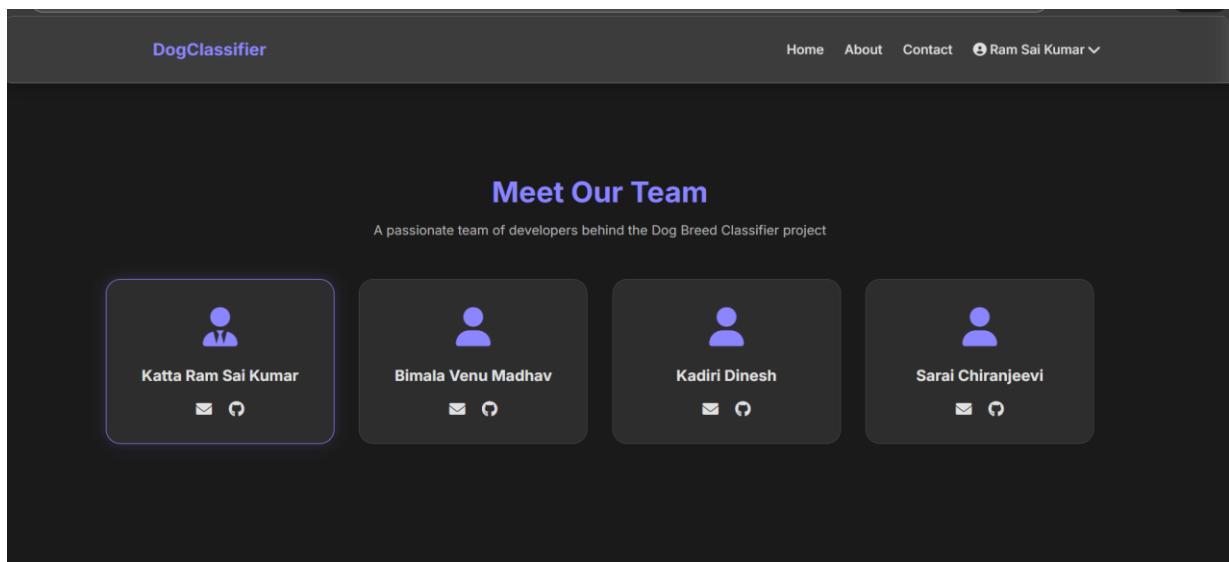
- index.html
- login.html
- signup.html

and save them in templates folder.

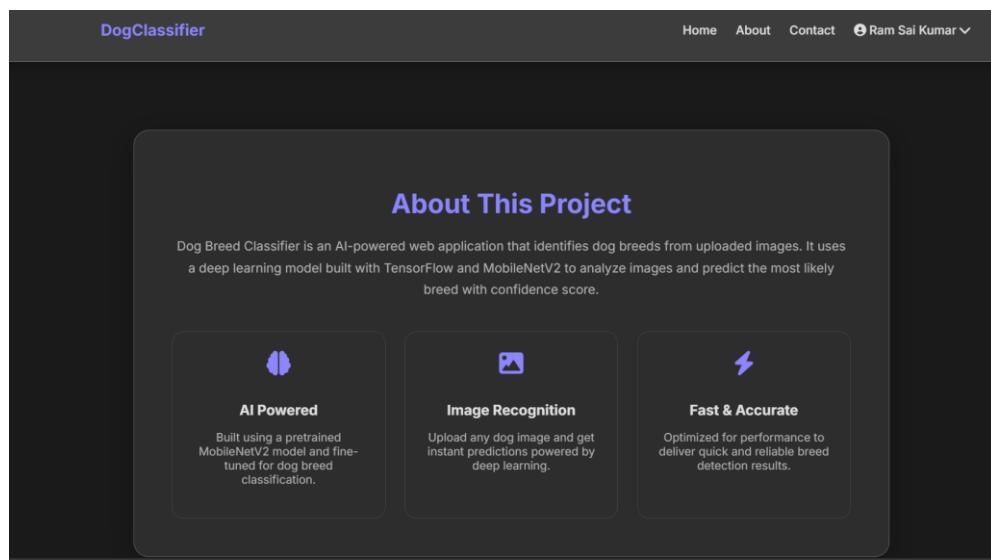
Let's see how our home.html page looks like:



Let's see how our contact.html page looks like:



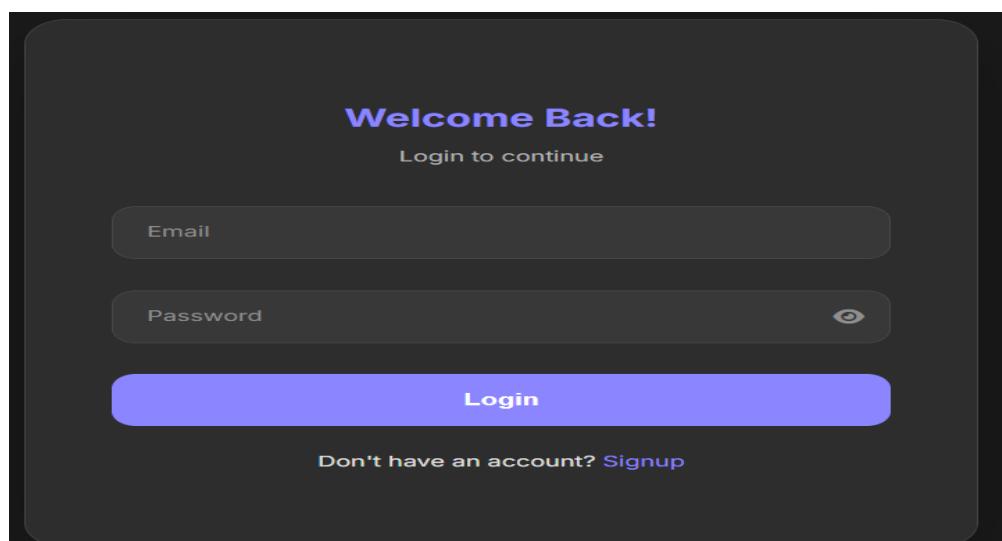
Lets look how our about.html file looks like:



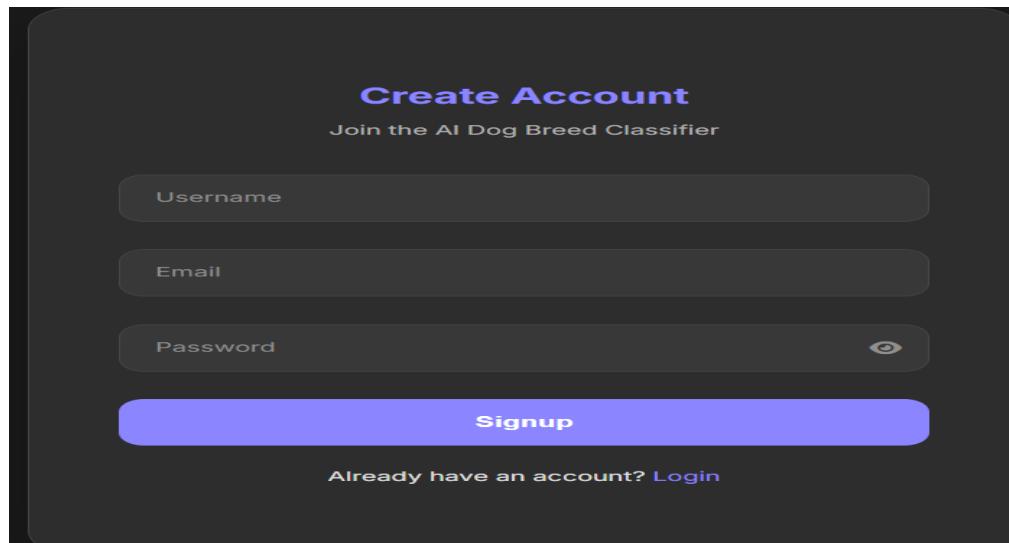
Lets Look how our index.html page looks like:



Lets Look how our login.html page looks like:



Lets Look how our signup.html page looks like:



## Activity 2: Build Python Code

Develop the backend using Python and the Flask framework to handle image upload, model loading, and prediction processing.

### Importing Libraries

Required libraries such as Flask, TensorFlow/Keras, NumPy, and image preprocessing modules are imported to support model inference and web interaction.

```
import os
from flask import Flask, request, jsonify, render_template, redirect, url_for, session, flash
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow_hub as hub
from PIL import Image
import io
import base64
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
```

### Creating Flask Application and Loading Model

A Flask app is initialized, and the trained MobileNetV2 dog breed classification model is loaded using the `load_model()` method to enable real-time predictions.

```

app = Flask(__name__)

# === AUTH CONFIG (ADDED ONLY) === #
app.secret_key = "super_secret_key_change_this"
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///{} + os.path.join(BASE_DIR, "users.db")
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db = SQLAlchemy(app)

# === USER MODEL (ADDED ONLY) === #
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(120), nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(200), nullable=False)

with app.app_context():
    db.create_all()

```

## Routing to HTML Page

Flask routes are defined to connect backend logic with frontend pages, allowing users to upload images and receive prediction results on the web interface.

```

# === ROUTES === #
@app.route("/")
def home():
    return render_template("home.html")

@app.route("/index")
def index():
    if "user_id" not in session:
        flash("Please login to continue", "error")
    return redirect(url_for("home"))

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

@app.route("/contact")
def contact():

```

## Run the Application

Finally, the Flask server is executed locally or deployed online, enabling users to access the dog breed prediction system through a browser.

```

venv) PS C:\Users\katta\Dog-Breed-Classification> python app.py
o enable the following instructions: SSE3 SSE4.1 SSE4.2 AVX AVX2 AVX512F AVX512_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
odel loaded successfully.
* Serving Flask app 'app'
* Debug mode: off
NFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI serv
r instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://192.168.1.8:5000
NFO:werkzeug:Press CTRL+C to quit
NFO:werkzeug:127.0.0.1 - - [17/Feb/2026 18:10:47] "GET / HTTP/1.1" 200 -

```

## Dog Breed Classifier



Upload a dog image and instantly discover its breed using  
AI

[Click here](#)