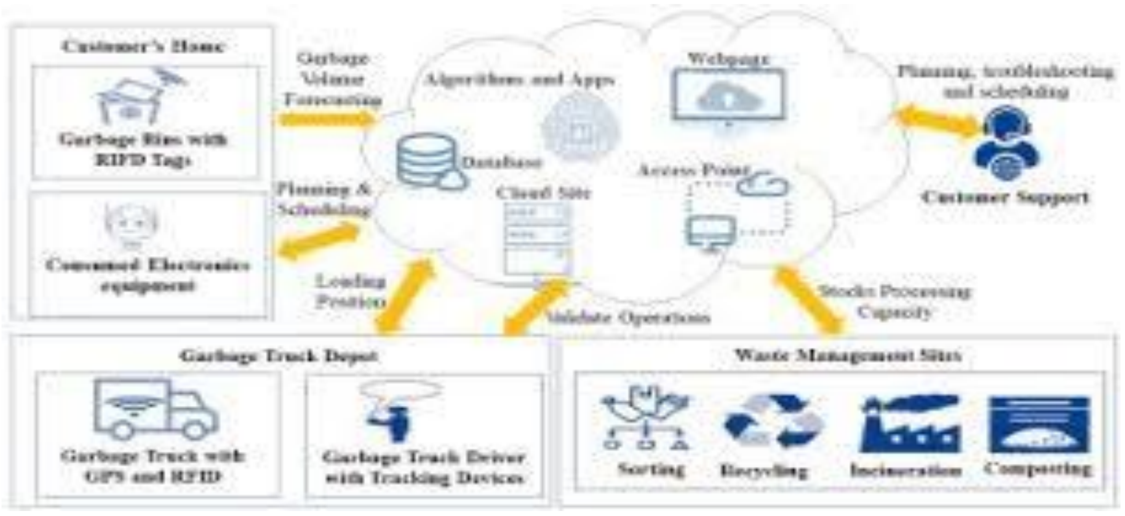
# PROJECT TITLE: CleanTech: Transforming Waste Management With Transfer Learning

**TEAM LEADER: Patan Middahath Banu**

**TEAM MEMBER: Kokku Rani**

**TEAM MEMBER: Kodamala Vasu**

**TEAM MEMBER: Muppagowni Pushpa Latha**


Customer's Home
Garbage Bins with RFID Tags
Consumed Electronics equipment
Garbage Volume Forecasting
Planning & Scheduling
Loading Position
Algorithms and Apps
Database
Cloud Site
Webpage
Access Point
Validate Operations
Planning, troubleshooting and scheduling
Customer Support
Stocks Processing Capacity
Garbage Truck Depot
Garbage Truck with GPS and RFID
Garbage Truck Driver with Tracking Devices
Waste Management Sites
Sorting
Recycling
Incineration
Composting

## INTRODUCTION:

The global waste crisis is intensifying with urbanization and consumerism, leading to overflowing landfills, inefficient recycling systems, and increased environmental degradation. Traditional waste management systems are often manual, error-prone, and lack the technological sophistication required for efficient segregation and processing.

CleanTech addresses this challenge by leveraging transfer learning, an advanced machine learning approach, to automate and optimize the waste identification and classification process. This project aims to build a scalable AI-driven solution that can be integrated with smart bins and municipal infrastructure, enabling cleaner cities and smarter waste cycles.

## 2. Problem Statement:

Manual waste sorting is labor-intensive, inconsistent, and often exposes workers to hazardous materials. While some waste management facilities use automated systems, they require large volumes of labeled data and substantial computational resources to train models from scratch. There is a need for a cost-effective, intelligent solution that can:

Accurately classify various types of waste (plastic, metal, organic, e-waste, etc.)

Function efficiently even with limited labeled datasets Integrate with existing smart infrastructure

## 3. Objective:

To develop an intelligent waste classification and management system using transfer learning that:

Identifies and classifies waste in real-time using visual inputs Supports edge deployment with smart bins or cameras enhances recycling, reduces contamination, and supports smart city goals.

## 4. Methodology:

a. Data Collection & Preprocessing:

Collect a diverse dataset of labeled waste images (open datasets + custom captures)

Categories include: plastic, paper, metal, glass, organic, hazardous, and e-waste

Perform data augmentation (rotation, flipping, lighting changes) to enhance model robustnes

b. Transfer Learning Implementation:

Utilize pre-trained CNN models (e.g., ResNet50, MobileNetV2, EfficientNet) Fine-tune the model on the waste classification dataset use techniques like freezing initial layers and retraining top layers to reduce training time

c. Model Evaluation:

Evaluate performance using metrics: accuracy, precision, recall, F1-score confusion matrix to assess misclassifications and optimize further

d. Integration with Hardware (Smart Bin Prototype):

Deploy model on edge devices (Raspberry Pi + camera module).Smart bin prototype automatically classifies and sorts waste into bins.Real-time alert and logging via IoT dashboard

e. Dashboard & Analytics:

Create a web/mobile interface using Streamlit or Flask.Monitor waste statistics, bin fill levels, and generate insights for authorities

## 5. Technologies Used:

Python, TensorFlow / PyTorch.OpenCV for image handling.Transfer Learning Models: ResNet, MobileNet, EfficientNet.YOLOv5 / Faster R-CNN for object detection (optional enhancement).Raspberry Pi, Arduino, camera module for smart bin.Streamlit, Flask, Firebase for dashboard and data backend.MQTT / Node-RED for IoT integration

## 6. Key Features:

High-accuracy waste classification with minimal data.Real-time sorting via camera input.Lightweight model for edge deployment.Scalable across urban and rural applications.Dashboard for analytics and reporting

## 7. Expected Outcomes:

Improved waste segregation accuracy by over 90%.Reduction in recyclable contamination.Enhanced public sanitation and reduced landfill pressure.Real-time insights for urban waste planners.Framework for future AI-powered environmental innovations

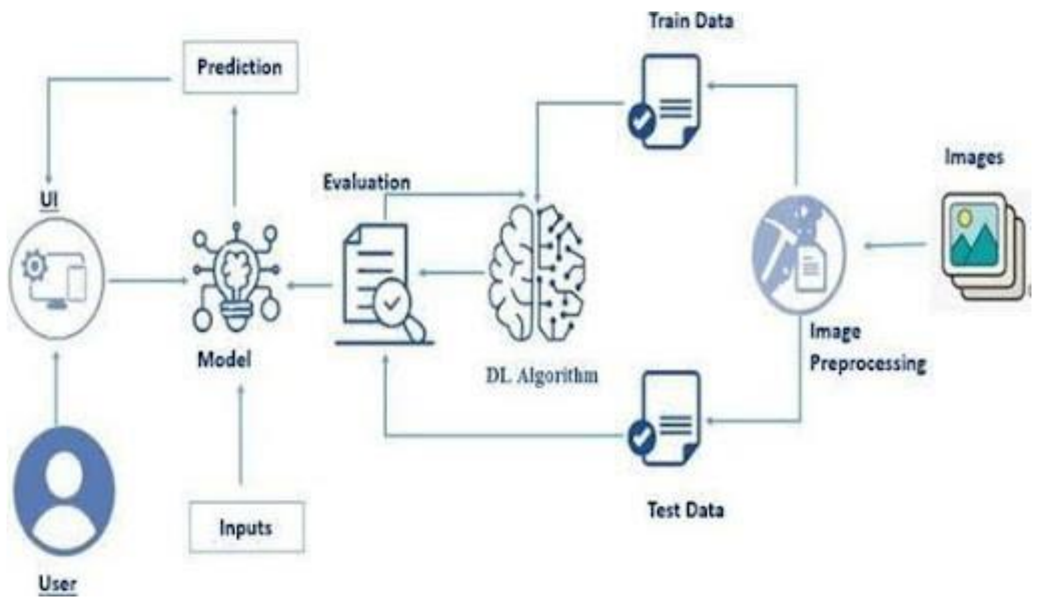
## 8. Future Scope:

Expansion to audio/sensor-based waste detection (e.g., weight, smell sensors)

Federated learning for privacy-preserving model improvements across locations integration with robotic arms for full automation AI recommendations for optimal waste routing and recycling strategies.

## 9. Impact and Innovation:

CleanTech reimagines waste management through AI-powered automation and data-driven decision-making. By minimizing manual effort, maximizing accuracy, and enabling smart city integration, this project has the potential to become a cornerstone of sustainable urban living and circular economy ecosystems.



# Prerequisites

- To complete this project, you must require the following software, concepts, and packages
- Anaconda Navigator:
- Refer to the link below to download Anaconda Navigator
- Python packages:
- Open anaconda prompt as administrator
- Type “pip install numpy” and click enter. • Type “pip install pandas” and click enter.
- Type “pip install scikit-learn” and click enter.

- Type ”pip install matplotlib” and click enter.
- Type ”pip install scipy” and click enter.
- Type ”pip install seaborn” and click enter.
- Type ”pip install tenserflow” and click enter.
- Type “pip install Flask” and click enter.

## Project Objectives

By the end of this project, you will:

- No fundamental concepts and techniques used for Deep Learning.
- Gain a broad understanding of data.
- Have knowledge of pre-processing the data/transformation techniques on outliers and some visualization concepts.

## 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 the flask application.
- Once the model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

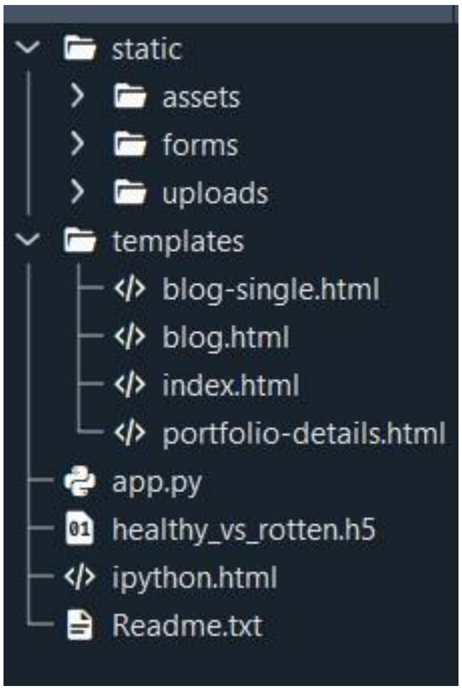
- Data Collection: Collect or download the dataset that you want to train.
- Data pre-processing
- Data Augmentation
- Splitting data into train and test

Model building

- Import the model-building libraries
- Initializing the model
- Training and testing the model
- Evaluating the performance of the model
- Save the model Application Building
- Create an HTML file
- Build python code

## Project Structure

Create the Project folder which contains files as shown below



```
static
    assets
    forms
    uploads
templates
    blog-single.html
    blog.html
    index.html
    portfolio-details.html
app.py
healthy_vs_rotten.h5
ipython.html
Readme.txt
```

- We are building a Flask application with HTML pages stored in the templates folder and a Python script app.py for scripting.
- Vgg16.h5 is our saved model. Further, we will use this model for flask integration.

## Data Collection And Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

## Collect The Dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project, we have used 3 classes of biodegradable, recyclable and trash images data. This data is downloaded from kaggle.com or can be connected by using API. Please refer to the link given below to download the dataset.

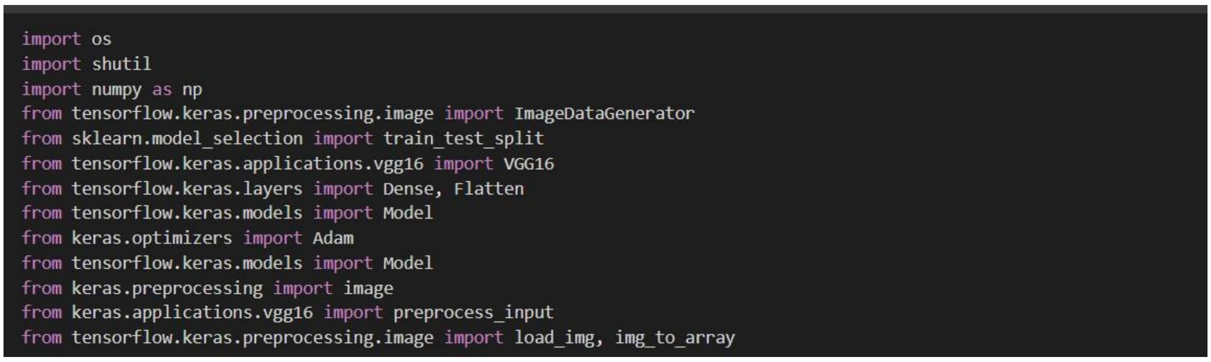
Link: Dataset

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

**Note: There are several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.**

## Activity 1.1: Importing the libraries:

Import the necessary libraries as shown in the image.

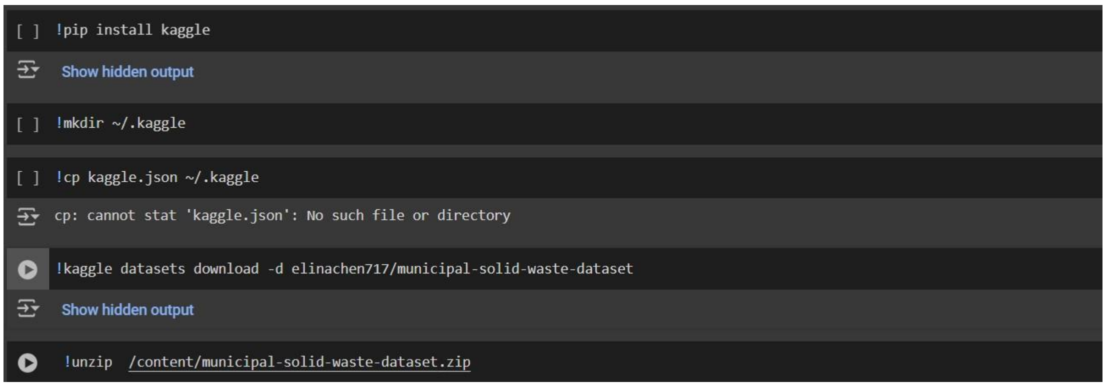


```
import os
import shutil
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from keras.optimizers import Adam
from tensorflow.keras.models import Model
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

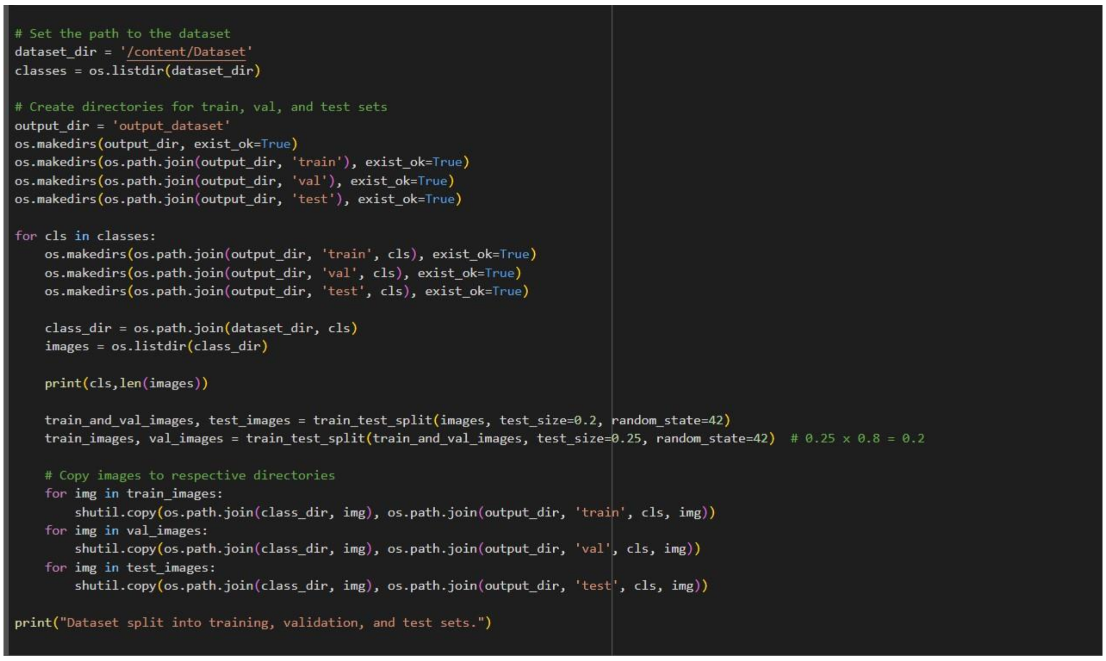
## Activity 1.2: Read the Dataset:

- Our dataset format might be in .csv, excel files, .txt, .json, or zip files, etc. We can read the dataset with the help of pandas.

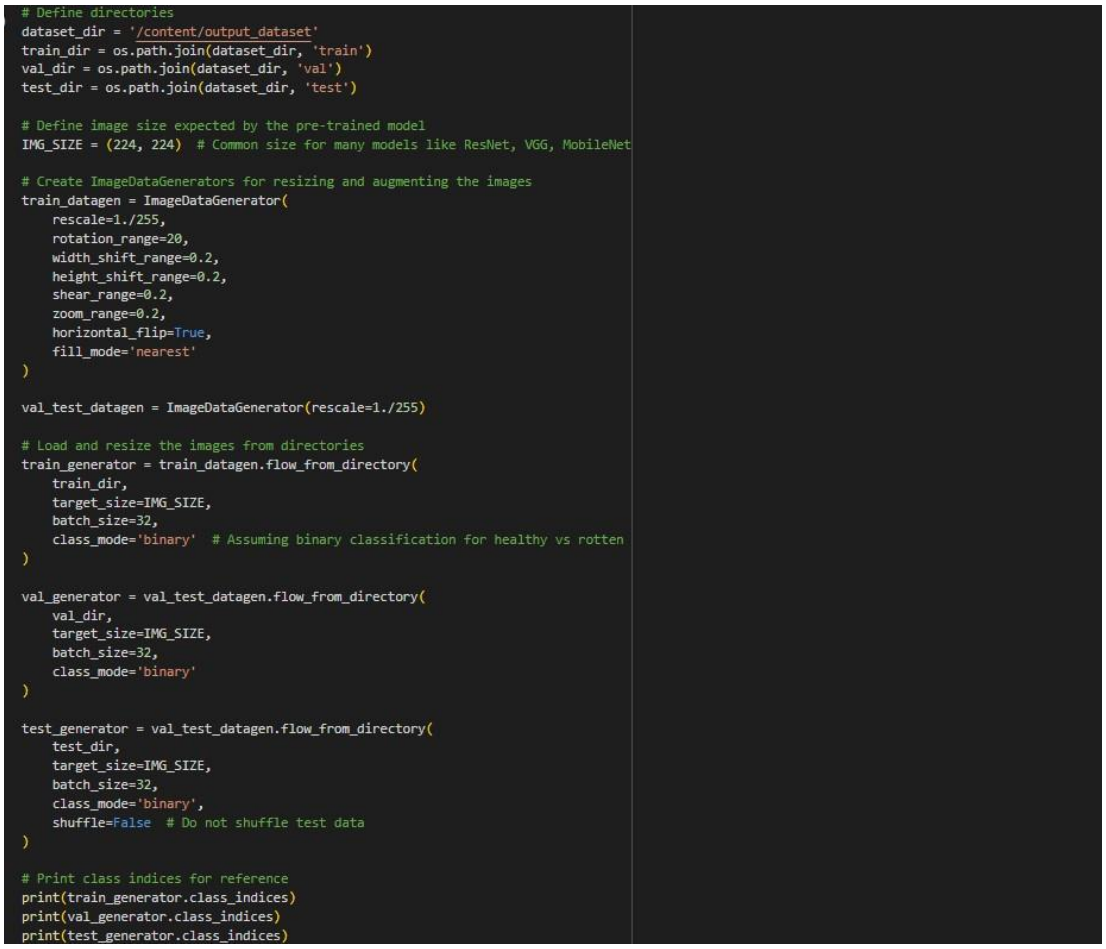
  At first, unzip the data and convert it into a pandas data frame.



```
[ ] !pip install kaggle
    Show hidden output

[ ] !mkdir ~/.kaggle

[ ] !cp kaggle.json ~/.kaggle
    cp: cannot stat 'kaggle.json': No such file or directory

    !kaggle datasets download -d elinachen717/municipal-solid-waste-dataset
    Show hidden output

    !unzip  /content/municipal-solid-waste-dataset.zip
```



```python
# Set the path to the dataset
dataset_dir = '/content/Dataset'
classes = os.listdir(dataset_dir)

# Create directories for train, val, and test sets
output_dir = 'output_dataset'
os.makedirs(output_dir, exist_ok=True)
os.makedirs(os.path.join(output_dir, 'train'), exist_ok=True)
os.makedirs(os.path.join(output_dir, 'val'), exist_ok=True)
os.makedirs(os.path.join(output_dir, 'test'), exist_ok=True)

for cls in classes:
    os.makedirs(os.path.join(output_dir, 'train', cls), exist_ok=True)
    os.makedirs(os.path.join(output_dir, 'val', cls), exist_ok=True)
    os.makedirs(os.path.join(output_dir, 'test', cls), exist_ok=True)

    class_dir = os.path.join(dataset_dir, cls)
    images = os.listdir(class_dir)

    print(cls,len(images))

    train_and_val_images, test_images = train_test_split(images, test_size=0.2, random_state=42)
    train_images, val_images = train_test_split(train_and_val_images, test_size=0.25, random_state=42)  # 0.25 x 0.8 = 0.2

    # Copy images to respective directories
    for img in train_images:
        shutil.copy(os.path.join(class_dir, img), os.path.join(output_dir, 'train', cls, img))
    for img in val_images:
        shutil.copy(os.path.join(class_dir, img), os.path.join(output_dir, 'val', cls, img))
    for img in test_images:
        shutil.copy(os.path.join(class_dir, img), os.path.join(output_dir, 'test', cls, img))

print("Dataset split into training, validation, and test sets.")
```

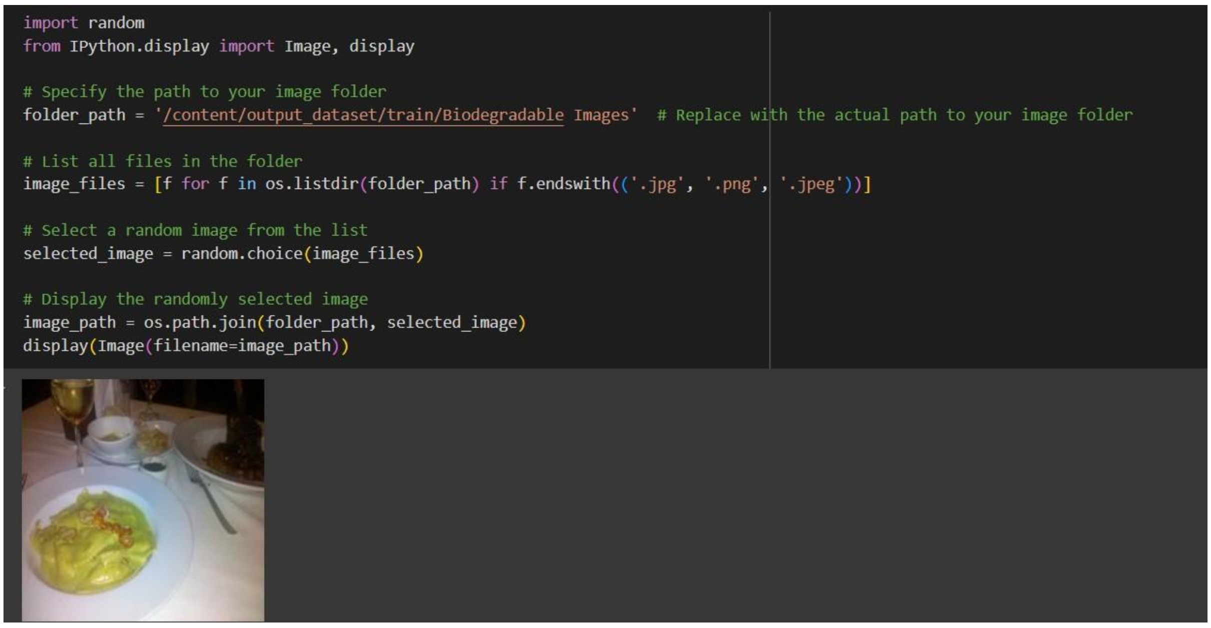


```python
# Define directories
dataset_dir = '/content/output_dataset'
train_dir = os.path.join(dataset_dir, 'train')
val_dir = os.path.join(dataset_dir, 'val')
test_dir = os.path.join(dataset_dir, 'test')

# Define image size expected by the pre-trained model
IMG_SIZE = (224, 224)  # Common size for many models like ResNet, VGG, MobileNet

# Create ImageDataGenerators for resizing and augmenting the images
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

val_test_datagen = ImageDataGenerator(rescale=1./255)

# Load and resize the images from directories
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary'  # Assuming binary classification for healthy vs rotten
)

val_generator = val_test_datagen.flow_from_directory(
    val_dir,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary'
)

test_generator = val_test_datagen.flow_from_directory(
    test_dir,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary',
    shuffle=False  # Do not shuffle test data
)

# Print class indices for reference
print(train_generator.class_indices)
print(val_generator.class_indices)
print(test_generator.class_indices)
```

## Data Visualization

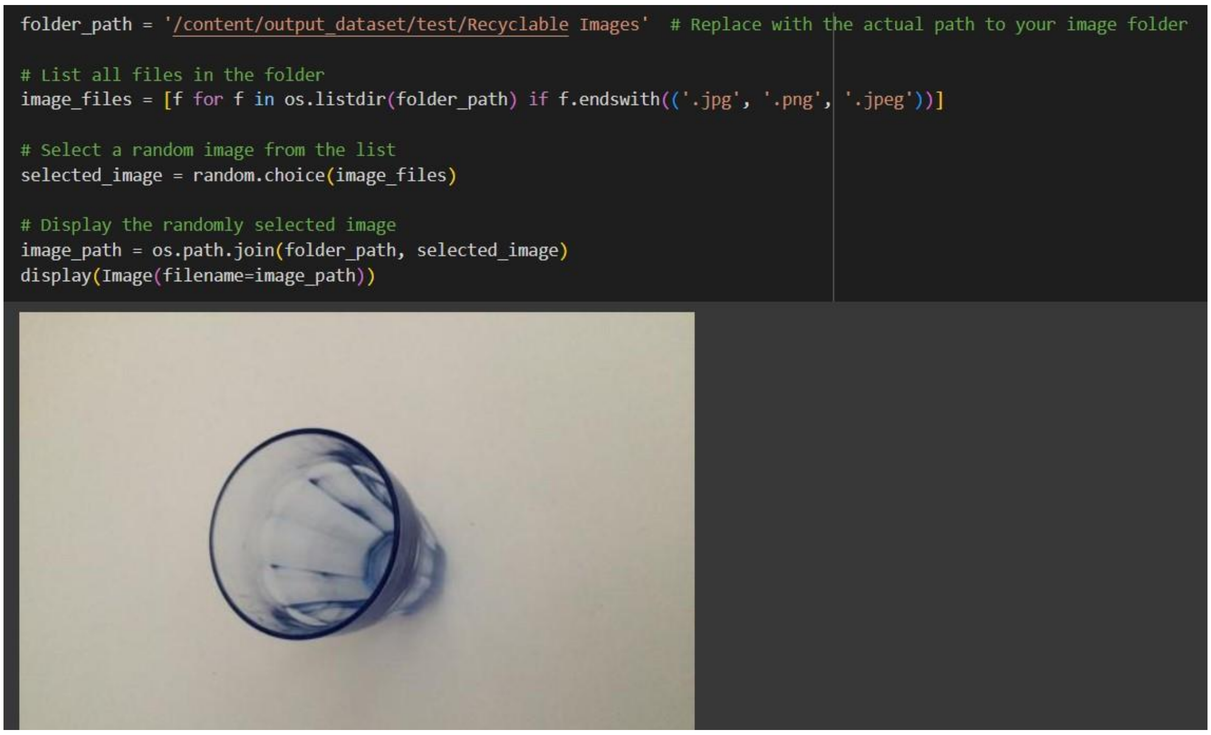
The provided Python code imports necessary libraries and modules for image manipulation.

It selects a random image file from a specified folder path. Then, it displays the randomly selected image using IPython's Image module. This code is useful for showcasing random images from a directory for various purposes like data exploration or testing image processing algorithms.



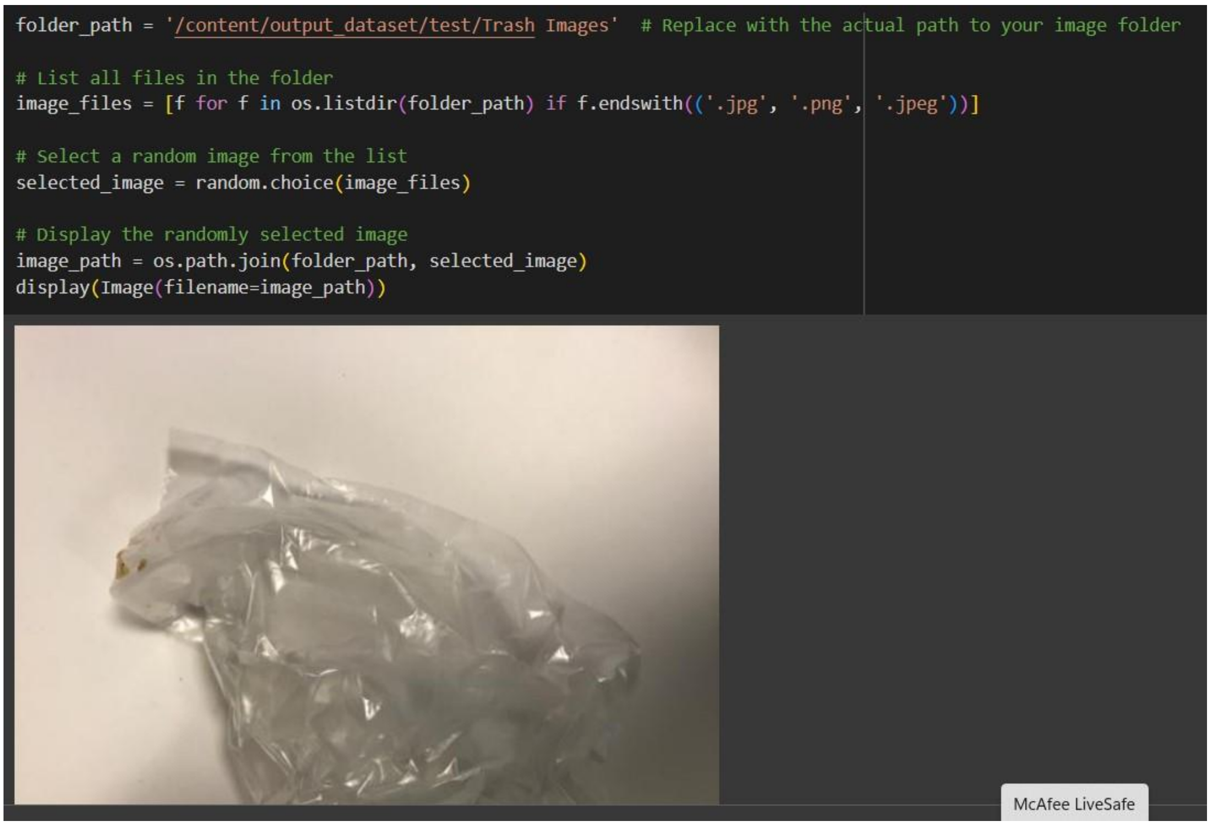
```python
import random
from IPython.display import Image, display

# Specify the path to your image folder
folder_path = '/content/output_dataset/train/Biodegradable Images'  # Replace with the actual path to your image folder

# List all files in the folder
image_files = [f for f in os.listdir(folder_path) if f.endswith(('.jpg', '.png', '.jpeg'))]

# Select a random image from the list
selected_image = random.choice(image_files)

# Display the randomly selected image
image_path = os.path.join(folder_path, selected_image)
display(Image(filename=image_path))
```

In the above code, I used class biodegradable class for prediction, This code randomly selects an image file from a specified folder (folder_path) containing JPEG, PNG, or JPEG files, and then displays the selected image using IPython's display function. It utilizes Python's OS and random modules for file manipulation and random selection, respectively. And It has predicted correctly as biodegradable image.



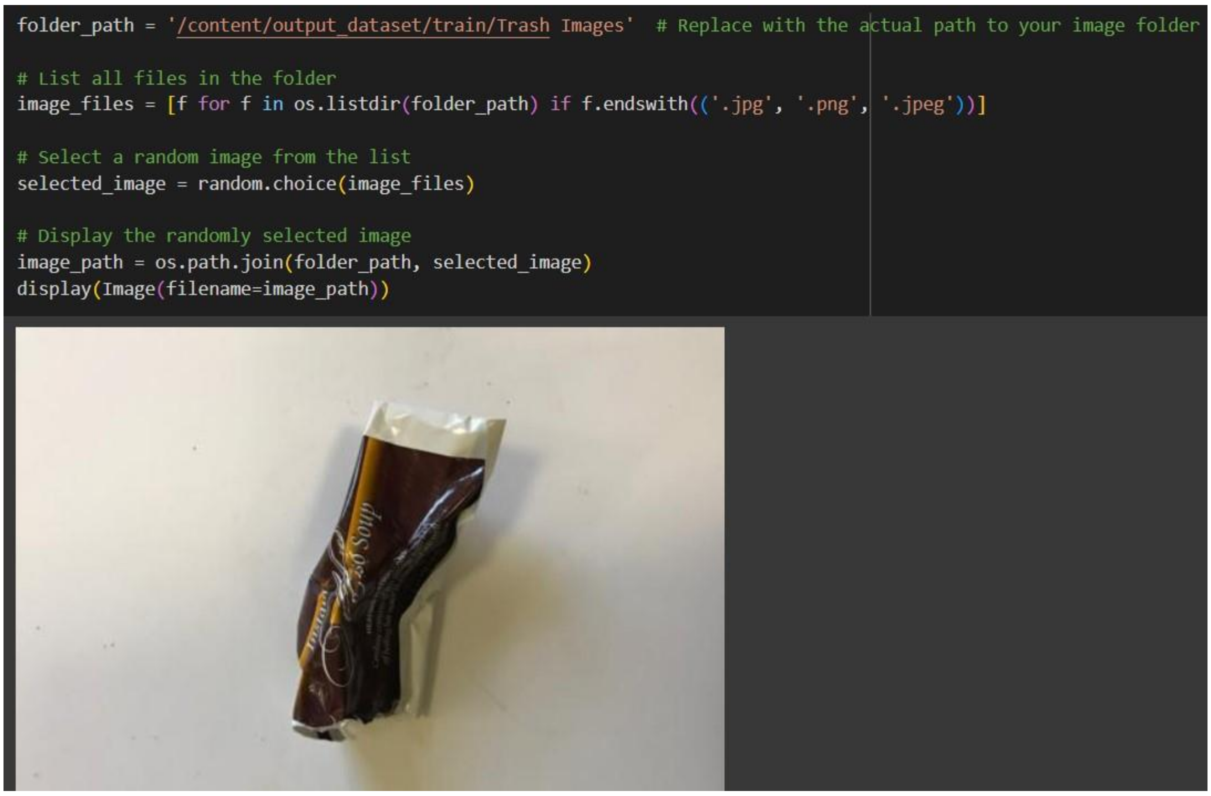
```
folder_path = '/content/output_dataset/test/Recyclable Images'  # Replace with the actual path to your image folder

# List all files in the folder
image_files = [f for f in os.listdir(folder_path) if f.endswith(('.jpg', '.png', '.jpeg'))]

# Select a random image from the list
selected_image = random.choice(image_files)

# Display the randomly selected image
image_path = os.path.join(folder_path, selected_image)
display(Image(filename=image_path))
```

In the above code, I used recyclable class for prediction, This code randomly selects an image file from a specified folder (folder_path) containing JPEG, PNG, or JPEG files, and then displays the selected image using IPython's display function. It utilizes Python's OS and random modules for file manipulation and random selection, respectively. And It has predicted correctly as recyclable image.



```
folder_path = '/content/output_dataset/test/Trash Images'  # Replace with the actual path to your image folder

# List all files in the folder
image_files = [f for f in os.listdir(folder_path) if f.endswith(('.jpg', '.png', '.jpeg'))]

# Select a random image from the list
selected_image = random.choice(image_files)

# Display the randomly selected image
image_path = os.path.join(folder_path, selected_image)
display(Image(filename=image_path))
```

In the above code, I used  trash class  for prediction, This code randomly selects an image file from a specified folder (folder_path) containing JPEG, PNG, or JPEG files, and then displays the selected image using IPython's display function. It utilizes Python's OS and random modules for file manipulation and random selection, respectively. And It has predicted correctly as trash image.


```
folder_path = '/content/output_dataset/train/Trash Images'  # Replace with the actual path to your image folder

# List all files in the folder
image_files = [f for f in os.listdir(folder_path) if f.endswith(('.jpg', '.png', '.jpeg'))]

# Select a random image from the list
selected_image = random.choice(image_files)

# Display the randomly selected image
image_path = os.path.join(folder_path, selected_image)
display(Image(filename=image_path))
```

In the above code, I used class  for prediction, This code randomly selects an image file from a specified folder (folder_path) containing JPEG, PNG, or JPEG files, and then displays the selected image using IPython's display function. It utilizes Python's OS and random modules for file manipulation and random selection, respectively. And It has predicted correctly as trash image.

Data Augmentation

Data augmentation is a technique commonly employed in machine learning, particularly in computer vision tasks such as image classification, including projects like the healthy vs rotten Classification  in fruits and vegetables. The primary objective of data augmentation is to artificially expand the size of the training dataset by applying various transformations to the existing images, thereby increasing the diversity and robustness of the data available for model training. This approach is particularly beneficial when working with limited labeled data.

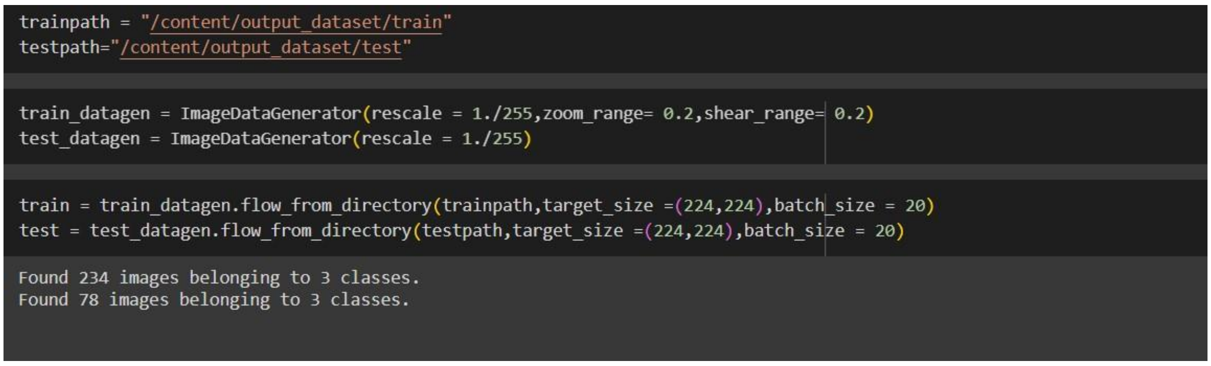
In the context of the 28  class Classification, data augmentation can involve applying transformations such as rotation, scaling, flipping, and changes in brightness or contrast to the original images of fossils. These transformations help the model generalize better to variations and potential distortions present in real-world images, enhancing its ability to accurately classify unseen data.

This is a crucial step but this data is already cropped from the augmented data so. this time it is skipped accuracy is not much affected but the training time increased.

Split Data And Model Building

Train-Test-Split:

In this project, we have already separated data for training and testing.



```
trainpath = "/content/output_dataset/train"
testpath="/content/output_dataset/test"
```

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

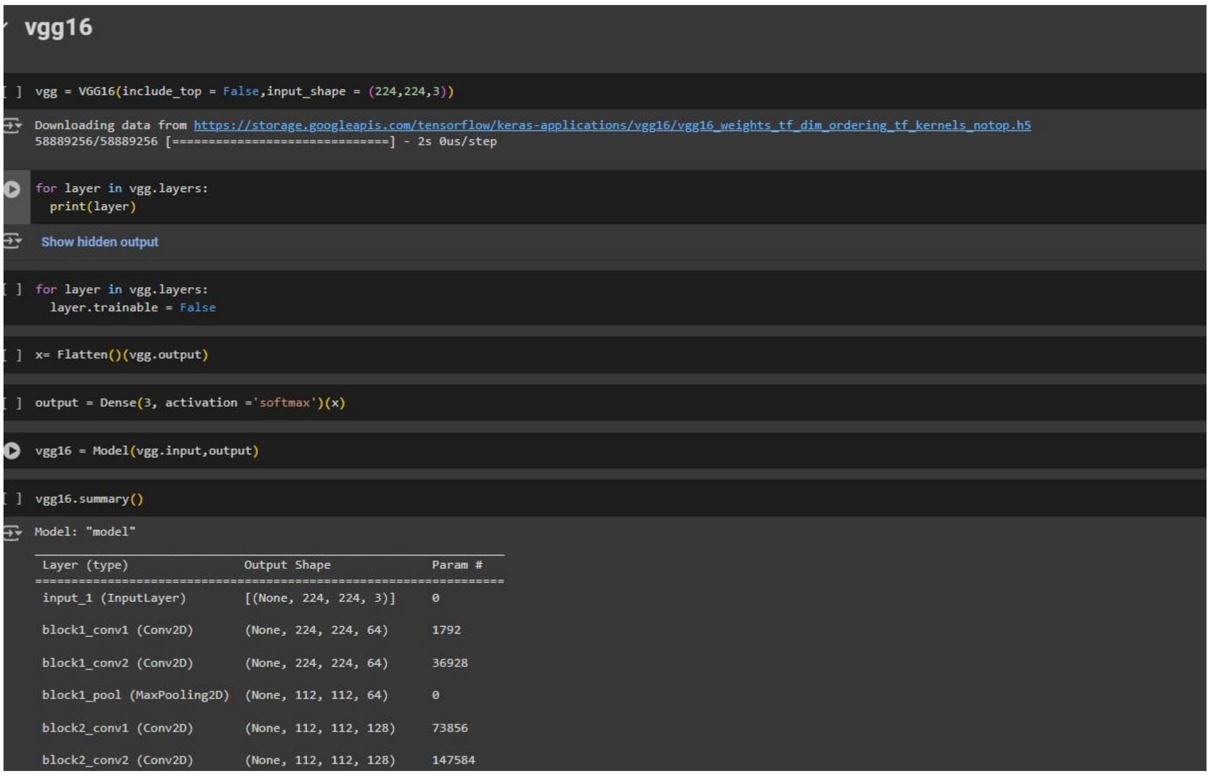
```
train = train_datagen.flow_from_directory(trainpath,target_size =(224,224),batch_size = 20)
test = test_datagen.flow_from_directory(testpath,target_size =(224,224),batch_size = 20)
```

```
Found 234 images belonging to 3 classes.
Found 78 images belonging to 3 classes.
```

## Model Building:

Vgg16 Transfer-Learning Model:

The VGG16-based neural network is created using a pre-trained VGG16 architecture with frozen weights. The model is built sequentially, incorporating the VGG16 base, a flattening layer, dropout for regularization, and a dense layer with SoftMax activation for classification into five categories. The model is compiled using the Adam optimizer and sparse categorical cross-entropy loss. During training, which spans 10 epochs, a generator is employed for the training data, and validation is conducted, incorporating call-backs such as Model Checkpoint and Early Stopping. The best-performing model is saved as "healthy_vs_rotten.h5 " for potential future use. The model summary provides an overview of the architecture, showcasing the layers and parameters involved.



vgg16

```
vgg = VGG16(include_top = False,input_shape = (224,224,3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 2s 0us/step
```

```
for layer in vgg.layers:
  print(layer)
```

```
for layer in vgg.layers:
  layer.trainable = False
```

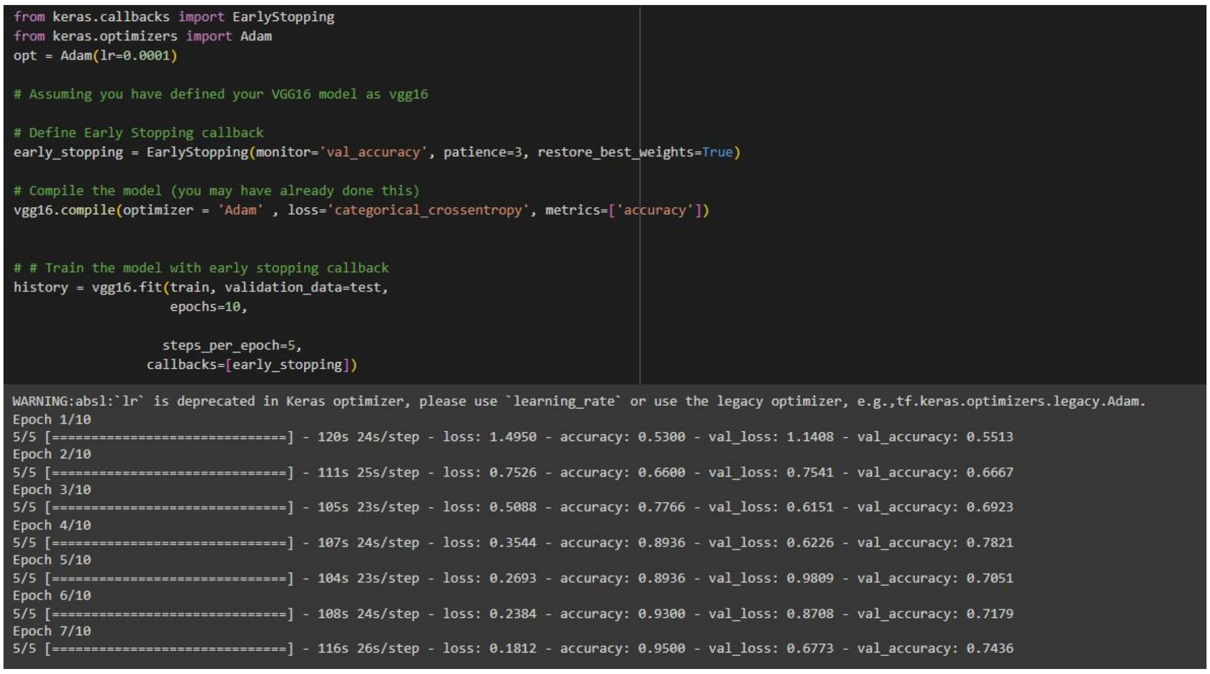
```
x= Flatten()(vgg.output)
```

```
output = Dense(3, activation ='softmax')(x)
```

```
vgg16 = Model(vgg.input,output)
```

```
vgg16.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584
```



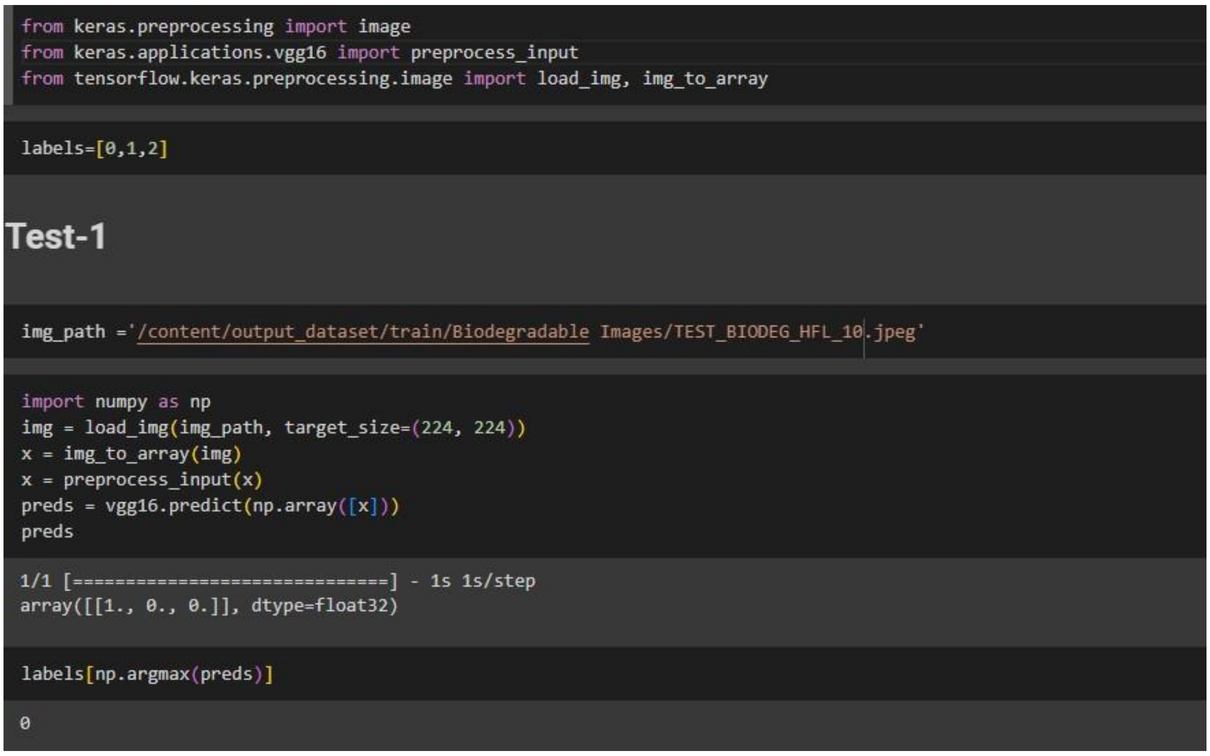
```
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
opt = Adam(lr=0.0001)

# Assuming you have defined your VGG16 model as vgg16

# Define Early Stopping callback
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, restore_best_weights=True)

# Compile the model (you may already done this)
vgg16.compile(optimizer = 'Adam' , loss='categorical_crossentropy', metrics=['accuracy'])


# # Train the model with early stopping callback
history = vgg16.fit(train, validation_data=test,
                    epochs=10,

                   steps_per_epoch=5,
                 callbacks=[early_stopping])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
Epoch 1/10
5/5 [==============================] - 120s 24s/step - loss: 1.4950 - accuracy: 0.5300 - val_loss: 1.1408 - val_accuracy: 0.5513
Epoch 2/10
5/5 [==============================] - 111s 25s/step - loss: 0.7526 - accuracy: 0.6600 - val_loss: 0.7541 - val_accuracy: 0.6667
Epoch 3/10
5/5 [==============================] - 105s 23s/step - loss: 0.5088 - accuracy: 0.7766 - val_loss: 0.6151 - val_accuracy: 0.6923
Epoch 4/10
5/5 [==============================] - 107s 24s/step - loss: 0.3544 - accuracy: 0.8936 - val_loss: 0.6226 - val_accuracy: 0.7821
Epoch 5/10
5/5 [==============================] - 104s 23s/step - loss: 0.2693 - accuracy: 0.8936 - val_loss: 0.9809 - val_accuracy: 0.7051
Epoch 6/10
5/5 [==============================] - 108s 24s/step - loss: 0.2384 - accuracy: 0.9300 - val_loss: 0.8708 - val_accuracy: 0.7179
Epoch 7/10
5/5 [==============================] - 116s 26s/step - loss: 0.1812 - accuracy: 0.9500 - val_loss: 0.6773 - val_accuracy: 0.7436
```

## Testing Model & Data Prediction

### Testing the model

Here we have tested with the Vgg16 Model With the help of the predict () function.



```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
labels=[0,1,2]
```

## Test-1

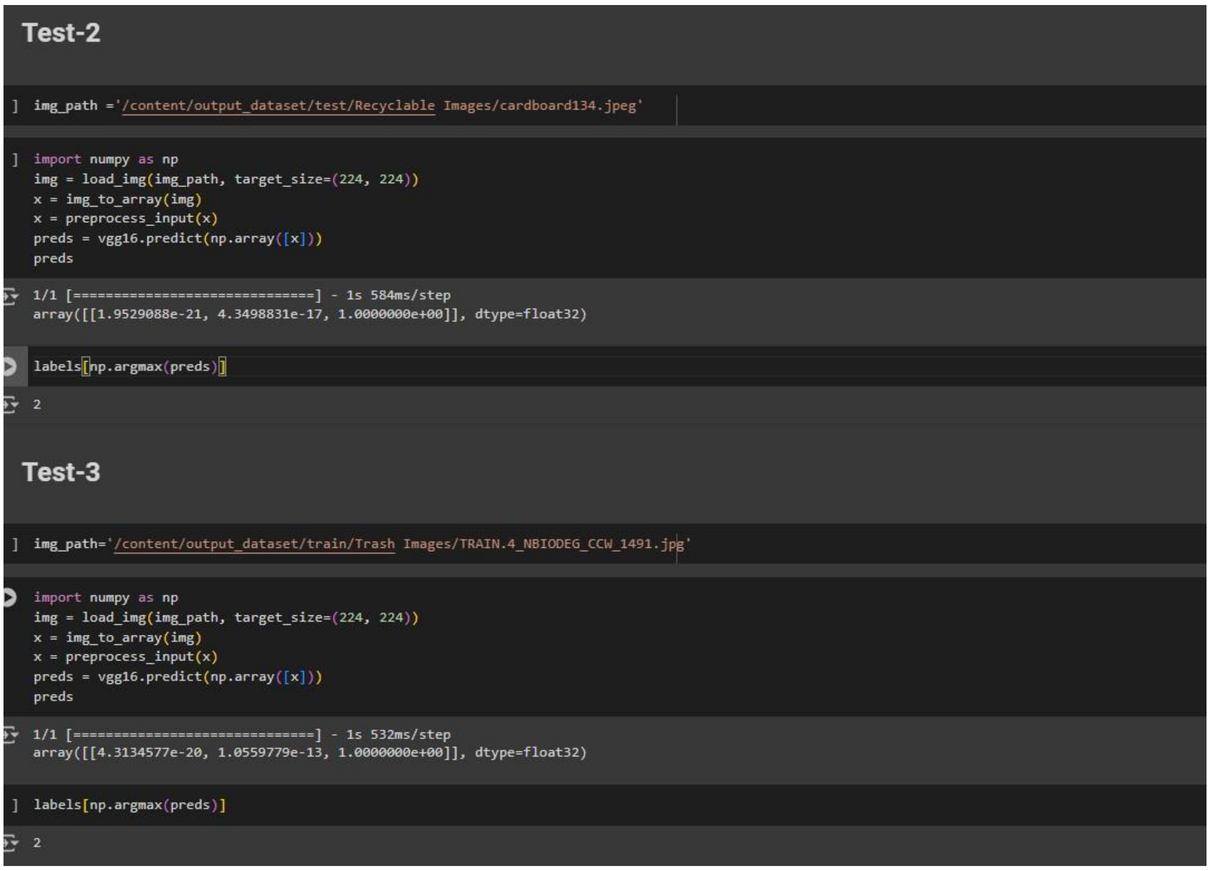
```
img_path ='/content/output_dataset/train/Biodegradable Images/TEST_BIODEG_HFL_10.jpeg'
```

```
import numpy as np
img = load_img(img_path, target_size=(224, 224))
x = img_to_array(img)
x = preprocess_input(x)
preds = vgg16.predict(np.array([x]))
preds
```

```
1/1 [==============================] - 1s 1s/step
array([[1., 0., 0.]], dtype=float32)
```

```
labels[np.argmax(preds)]
```

```
0
```



## Test-2

```
img_path ='/content/output_dataset/test/Recyclable Images/cardboard134.jpeg'
```

```
import numpy as np
img = load_img(img_path, target_size=(224, 224))
x = img_to_array(img)
x = preprocess_input(x)
preds = vgg16.predict(np.array([x]))
preds
```

```
1/1 [==============================] - 1s 584ms/step
array([[1.9529088e-21, 4.3498831e-17, 1.0000000e+00]], dtype=float32)
```

```
labels[np.argmax(preds)]
```

```
2
```

## Test-3

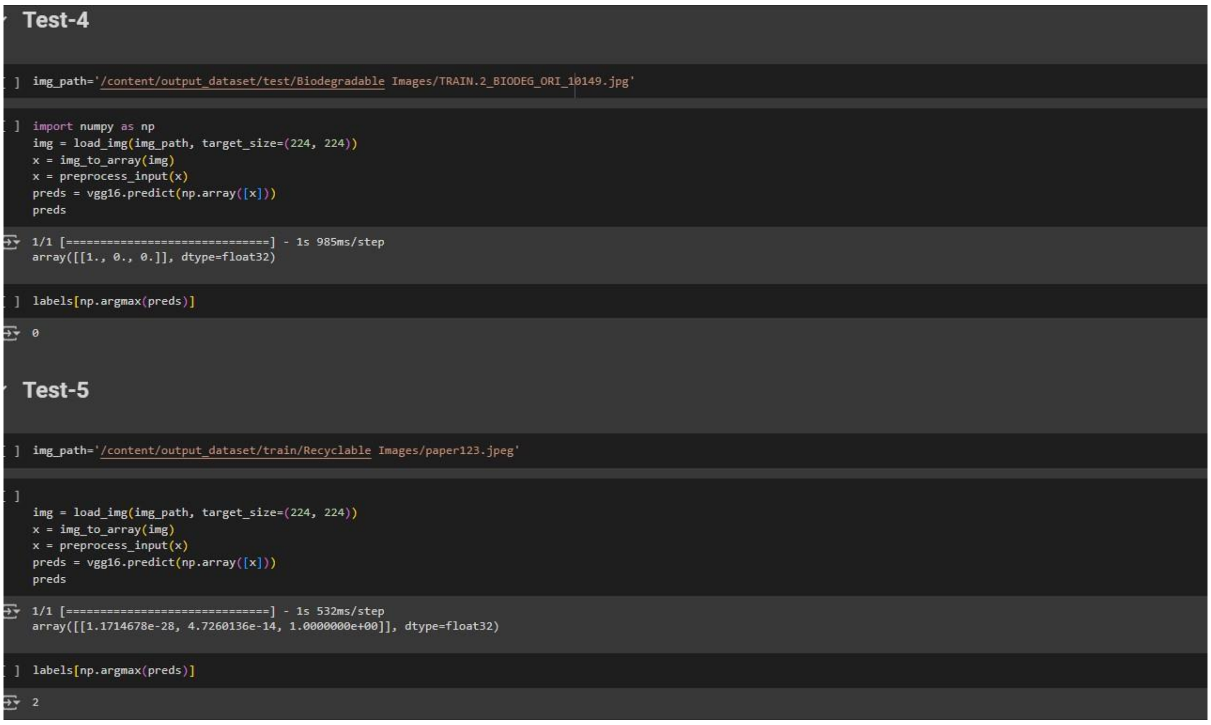
```
img_path='/content/output_dataset/train/Trash Images/TRAIN.4_NBIODEG_CCW_1491.jpg'
```

```
import numpy as np
img = load_img(img_path, target_size=(224, 224))
x = img_to_array(img)
x = preprocess_input(x)
preds = vgg16.predict(np.array([x]))
preds
```

```
1/1 [==============================] - 1s 532ms/step
array([[4.3134577e-20, 1.0559779e-13, 1.0000000e+00]], dtype=float32)
```

```
labels[np.argmax(preds)]
```

```
2
```



Test-4

```
img_path='/content/output_dataset/test/Biodegradable Images/TRAIN.2_BIODEG_ORI_10149.jpg'
```

```
import numpy as np
img = load_img(img_path, target_size=(224, 224))
x = img_to_array(img)
x = preprocess_input(x)
preds = vgg16.predict(np.array([x]))
preds
```

```
1/1 [==============================] - 1s 985ms/step
array([[1., 0., 0.]], dtype=float32)
```

```
labels[np.argmax(preds)]
```

```
0
```

Test-5

```
img_path='/content/output_dataset/train/Recyclable Images/paper123.jpeg'
```

```
img = load_img(img_path, target_size=(224, 224))
x = img_to_array(img)
x = preprocess_input(x)
preds = vgg16.predict(np.array([x]))
preds
```

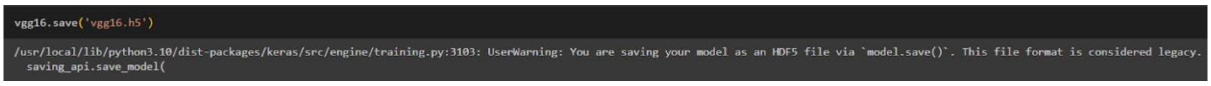
```
1/1 [==============================] - 1s 532ms/step
array([[1.1714678e-28, 4.7260136e-14, 1.0000000e+00]], dtype=float32)
```

```
labels[np.argmax(preds)]
```

```
2
```

## Saving The Model

Finally, we have chosen the best model now saving that model



```
vgg16.save('vgg16.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy.
  saving_api.save_model(
```

## Application Building

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

Building HTML Pages

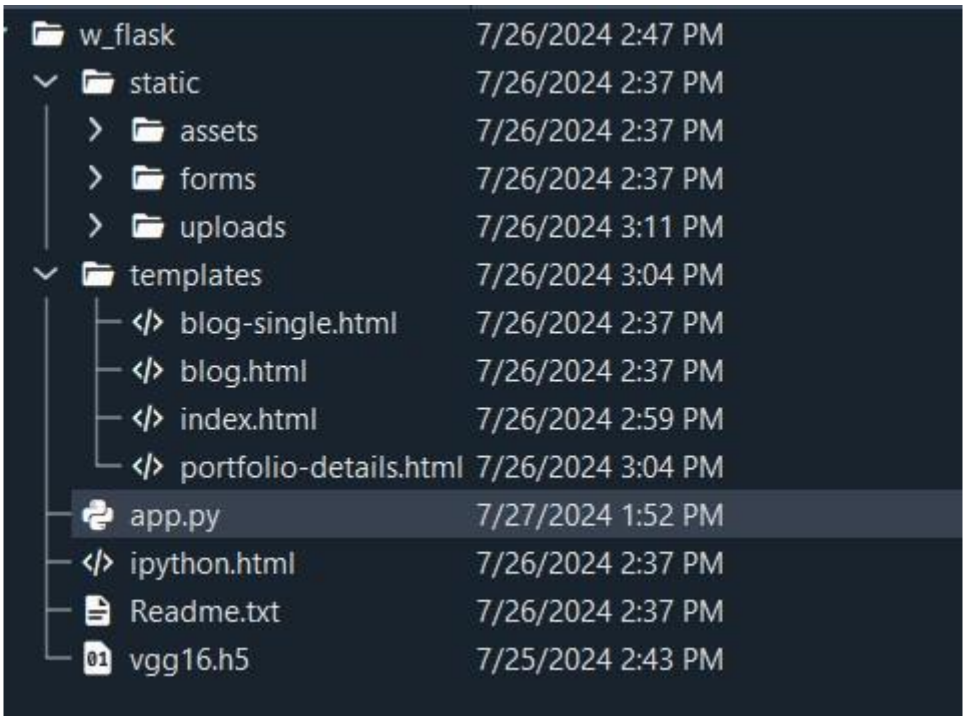
Building server-side script

## Building HTML Pages:

For this project create three HTML files namely

home.html result.html

Building HTML Pages:



```
w_flask                              7/26/2024 2:47 PM
  static                             7/26/2024 2:37 PM
    assets                           7/26/2024 2:37 PM
    forms                            7/26/2024 2:37 PM
    uploads                          7/26/2024 3:11 PM
  templates                          7/26/2024 3:04 PM
    blog-single.html                 7/26/2024 2:37 PM
    blog.html                        7/26/2024 2:37 PM
    index.html                       7/26/2024 2:59 PM
    portfolio-details.html           7/26/2024 3:04 PM
  app.py                             7/27/2024 1:52 PM
  ipython.html                       7/26/2024 2:37 PM
  Readme.txt                         7/26/2024 2:37 PM
  vgg16.h5                           7/25/2024 2:43 PM
```

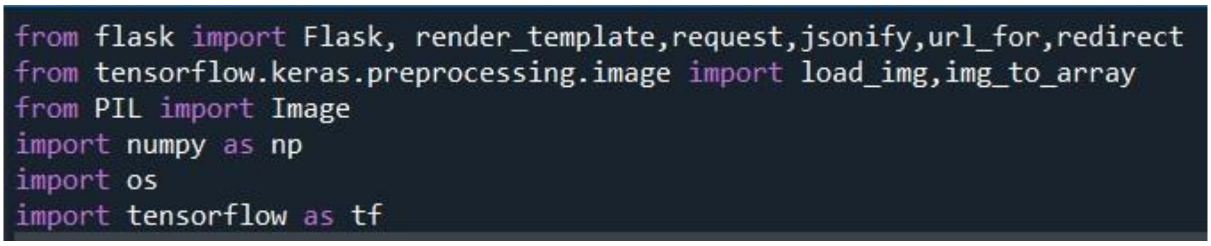
For this project create three HTML files namely

index.html portofolio_details.html

And save them in the templates folder.

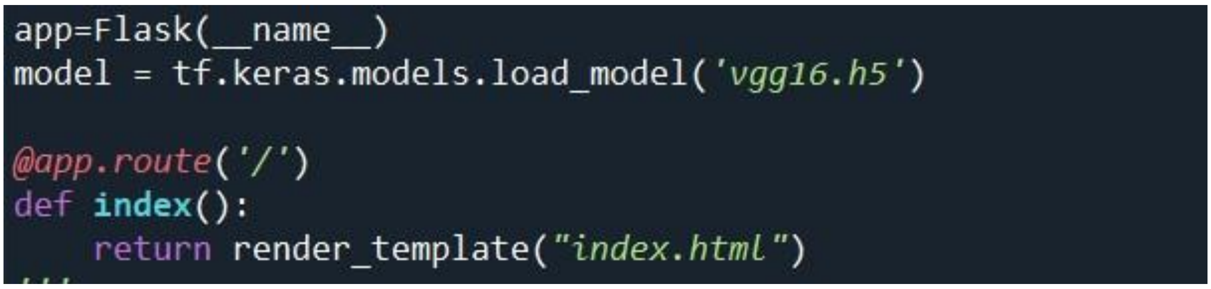
## Build Python Code:

Import the libraries



```python
from flask import Flask, render_template,request,jsonify,url_for,redirect
from tensorflow.keras.preprocessing.image import load_img,img_to_array
from PIL import Image
import numpy as np
import os
import tensorflow as tf
```

Load the saved model. Importing the Flask module in the project is mandatory. An object of the Flask class is our WSGI application. The Flask constructor takes the name of the current module (__name__) as argument.

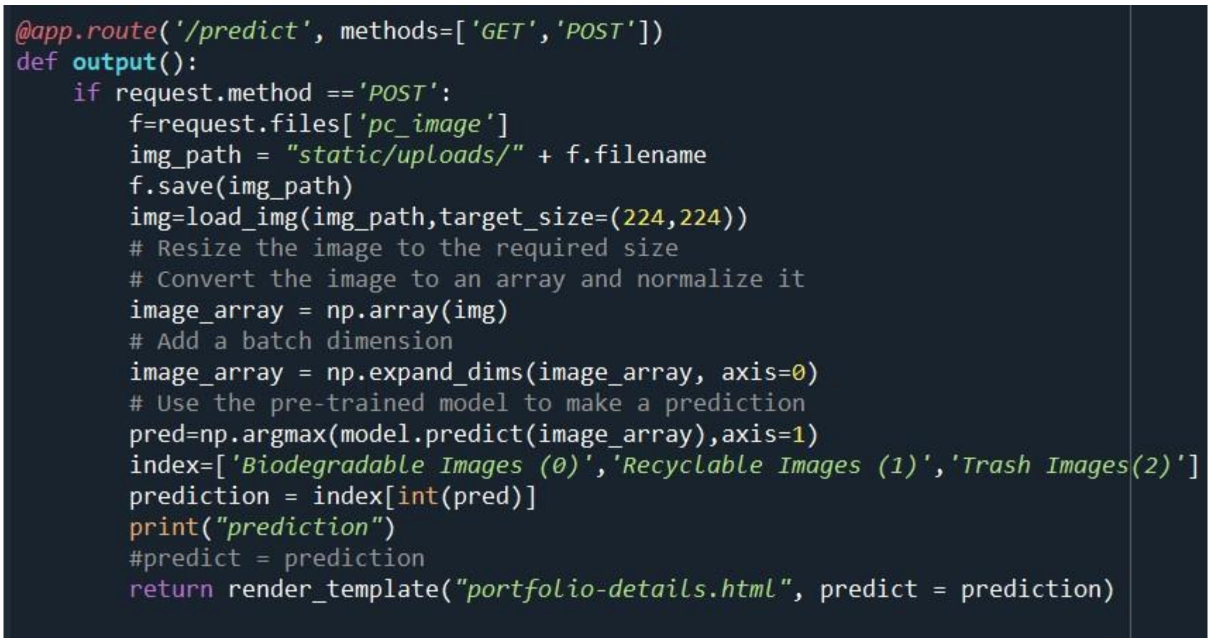


```
app=Flask(__name__)
model = tf.keras.models.load_model('vgg16.h5')

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

Here we will be using the declared constructor to route to the HTML page which we have created earlier.

In the above example, the '/' URL is bound with the index.html function. Hence, when the index page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

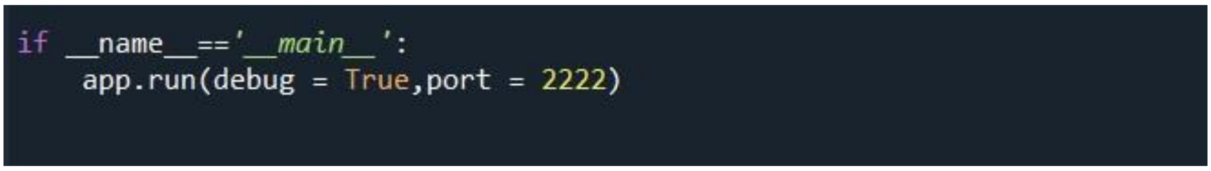
Retrieves the value from UI:



```
@app.route('/predict', methods=['GET','POST'])
def output():
    if request.method =='POST':
        f=request.files['pc_image']
        img_path = "static/uploads/" + f.filename
        f.save(img_path)
        img=load_img(img_path,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=['Biodegradable Images (0)','Recyclable Images (1)','Trash Images(2)']
        prediction = index[int(pred)]
        print("prediction")
        #predict = prediction
        return render_template("portfolio-details.html", predict = prediction)
```

Here we are routing our app to the output() function. This function retrieves all the values from the HTML page using a Post request. That is stored in an array. This array is passed to the model. Predict () function. This function returns the prediction. This prediction value will rendered to the text that we have mentioned in the output.html page earlier.

Main Function:



```
if __name__=='__main__':
    app.run(debug = True,port = 2222)
```

## Run The Web Application

Run the application

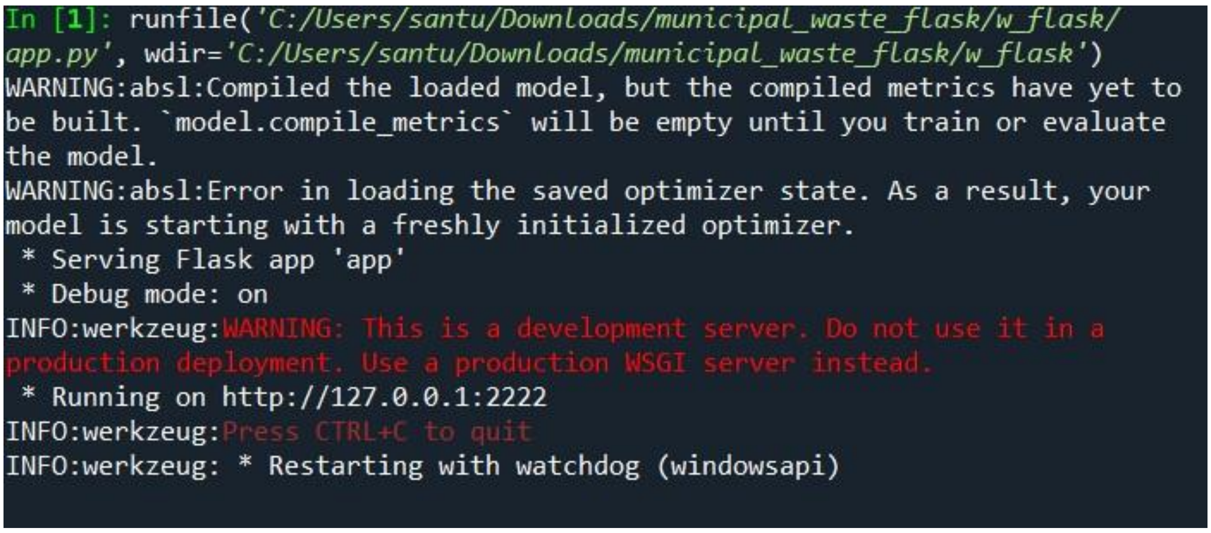
Open Anaconda prompt from the start menu

Navigate to the folder where your Python script is.

Now type the “app.py” command

Navigate to the local host where you can view your web page.

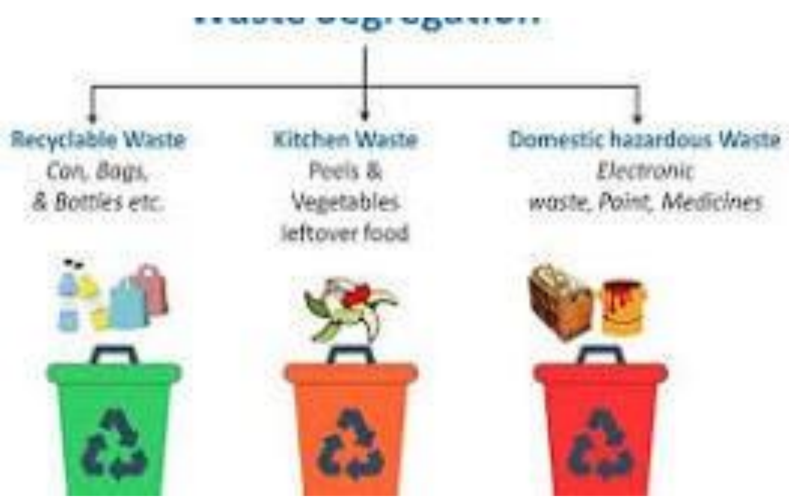
Click on the inspect button from the top right corner, enter the inputs, click on the predict button, and see the result/prediction on the web.



```
In [1]: runfile('C:/Users/santu/Downloads/municipal_waste_flask/w_flask/
app.py', wdir='C:/Users/santu/Downloads/municipal_waste_flask/w_flask')
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to
be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
WARNING:absl:Error in loading the saved optimizer state. As a result, your
model is starting with a freshly initialized optimizer.
 * Serving Flask app 'app'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:2222
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with watchdog (windowsapi)
```

UI Image preview:

Let’s see what our index.html page looks like:


Recyclable Waste
Can, Bags, & Bottles etc.
Kitchen Waste
Peels & Vegetables leftover food
Domestic hazardous Waste
Electronic waste, Paint, Medicines