

AI&ML Project Documentation

Introduction

Project Title:

CleanTech: Transforming Waste Management with Transfer Learning

Team Members :

Team ID : LTVIP2025TMID35830

- **Muddangala Nandini** – Data Collection and Cleaning
- **Thoti Nandu** – Model Design and Training
- **B Narasimha Naidu** – Evaluation and Optimization
- **B Nethan**– Deployment and Integration

Project Overview

Purpose:

The purpose of this project is to build a deep learning-based image classification model to automatically identify different types of municipal waste from images. By leveraging transfer learning with VGG16, this project addresses real-world challenges in waste segregation and environmental sustainability. It aims to provide an AI-powered solution that classifies waste into categories like biodegradable, recyclable, and trash, helping streamline waste management and promote eco-friendly practices.

Goals:

- Automate the classification of municipal waste using image-based AI.
- Improve the efficiency of waste sorting through smart technology.
- Raise awareness and assist organizations in managing waste responsibly.
- Deliver a lightweight, deployable model that can be used in real-world scenarios such as recycling centers, smart bins, and educational platforms.

Key Features:

- Accurate classification of waste images into three categories: biodegradable, recyclable, and trash.
- Transfer learning using a pre-trained VGG16 model for high performance on a limited dataset.
- Simple web interface for uploading and predicting waste categories.
- Flask-powered backend integrated with the trained deep learning model.
- Easy-to-use, responsive design requiring no technical expertise from users.

Prerequisites: To complete this project, the following tools, libraries, and knowledge are required:
Software Requirements:

To build and run the CleanTech: Waste Classification project, the following tools and environments were used:

- **Google Colab**
Used for model training, data preprocessing, and running deep learning code in a GPU-accelerated environment.
<https://colab.research.google.com>
- **Anaconda Navigator**
For managing local Python environments.
<https://www.anaconda.com/products/distribution>
- **Python 3.x (preferably 3.8 or above)**
- **Flask**
Used to develop the web application interface.
-

Python Libraries Used:

Install these using Anaconda Prompt or inside your Colab notebook:

```
pip install numpy  
pip install pandas  
pip install scikit-learn  
pip install matplotlib  
pip install seaborn  
pip install tensorflow  
pip install flask  
pip install pillow
```

Prior Knowledge Required

To effectively contribute to or understand this project, the following foundational topics and resources are essential:

- **Deep Learning Concepts**
Overview of deep learning and how neural networks operate.
<https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/>
- **Deep Learning Frameworks**
Detailed comparison between PyTorch and TensorFlow frameworks.
<https://www.knowledgehut.com/blog/data-science/pytorch-vs-tensorflow>

- **Transfer Learning**
Learn how to use pre-trained models like VGG for new tasks.
<https://towardsdatascience.com/a-demonstration-of-transfer-learning-of-vgg-convolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a>
- **VGG16 Architecture**
Explanation of VGG16 architecture used in this project.
<https://www.geeksforgeeks.org/vgg-16-cnn-model/>
- **Overfitting & Regularization**
Learn how to avoid overfitting in deep learning models using regularization.
<https://www.analyticsvidhya.com/blog/2021/07/prevent-overfitting-using-regularization-techniques/>
- **Deep Learning Optimizers**
Overview of optimizers like Adam, SGD, and RMSprop.
<https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/>
- **Flask Basics (for Deployment)**
Beginner-friendly tutorial to create web apps with Flask.
https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Objectives:

By the end of this project, you will:

- Know 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 :

1. **User uploads an image** through the web interface.
2. The image is **processed by a trained deep learning model** integrated with a Flask backend.
3. The **prediction result is displayed** on the UI.

Activities Overview:

1. Data Preparation:

Dataset collection, preprocessing, augmentation, and train-test splitting.

2. Model Building:

Import libraries, initialize and train the model, evaluate performance, and save the trained model.

3. Application Development:

Create the HTML UI and build the Flask-based Python backend for prediction.

Project Structure:

```
FlaskWasteApp/
├── app.py
├── vgg16_model.h5
├── static/
│   ├── css/
│   ├── images/
│   └── uploads/
├── templates/
│   ├── index.html
│   └── portfolio-details.html
├── venv/
├── README.txt
└── requirements.txt
```

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.

Municipal Solid Waste Dataset – Kaggle

<https://www.kaggle.com/datasets/elinachen717/municipal-solid-waste-dataset>

This dataset includes labeled images under categories such as **Recyclable**, **Biodegradable**, and **Trash**, making it ideal for training a waste classification model. The collected data was organized into separate folders by class and reviewed for quality before preprocessing.

```
[ ] !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
```

```

# 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.")

```

```

# 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=30,
    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 dataset was visually explored using Python. A script was used to randomly select and display images from each category using the IPython.display module. This helped ensure data consistency and quality before training. The code imports necessary libraries, picks a random image from a given folder (e.g., Trash, Biodegradable, or Recyclable), and displays it for visual inspection. This step supports better understanding of image distribution and content during early development.





Data Augmentation:

Data augmentation is a common technique used to increase dataset size and diversity by applying transformations like rotation, flipping, scaling, and brightness adjustment to images. It helps improve model generalization, especially when labeled data is limited.

In this project, augmentation was considered but ultimately skipped, as the dataset was already preprocessed and cropped. While skipping augmentation slightly increased training time, it did not significantly affect model accuracy.

Split Data and Model Building:

Train-Test-Split:

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

```
[ ] !ls waste_data/waste_data/Dataset

'Biodegradable Images' 'Recyclable Images' 'Trash Images'

#dataset_dir = '/content/waste_data/waste_data/Dataset'
import os
#dataset_dir = '/content/waste_data/waste_data/Dataset'
classes = os.listdir(dataset_dir)
print("Classes found:", classes)

Classes found: ['Recyclable Images', 'Trash Images', 'Biodegradable Images']

[ ] for cls in classes:
    class_dir = os.path.join(dataset_dir, cls)
    images = os.listdir(class_dir)
    print(f"Class: {cls} - {len(images)} images")

Class: Recyclable Images - 130 images
Class: Biodegradable Images - 130 images
Class: Trash Images - 130 images
```

Testing Model & Data Prediction

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

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

# Correct folder path
folder_path = "/content/waste_data/Dataset/Biodegradable Images"

# Get all image files (jpg, jpeg, png)
image_files = [f for f in os.listdir(folder_path) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]

# Choose a random image
selected_image = random.choice(image_files)

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



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

# Folder path for Trash Images
folder_path = '/content/waste_data/Dataset/Trash Images'

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

# 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))
```



Saving the model:

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

```
from google.colab import files
files.upload()
```

Choose Files vgg16_model.h5

- vgg16_model.h5(n/a) - 59852752 bytes, last modified: 6/24/2025 - 100% done

Saving vgg16_model.h5 to vgg16_model (1).h5

```
[ ] import os
os.listdir()
```

['.config',
'vgg16_model.keras',
'vgg16_model.h5',
'waste_data.zip',
'output_dataset',
'vgg16_model (1).h5',
'waste_data',
'sample_data']


```
[ ] os.rename("vgg16_model (1).keras", "vgg16_model.keras")
```

```
[ ] vgg16.save("vgg16_model.keras")
```

```
import os
os.listdir()
```

['.config',
'vgg16_model.keras',
'vgg16_model.h5',
'waste_data.zip',
'output_dataset',
'waste_data',
'sample_data']

```
[ ] from google.colab import files
files.download("vgg16_model.keras")
```

Downloading "vgg16_model.keras": 

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

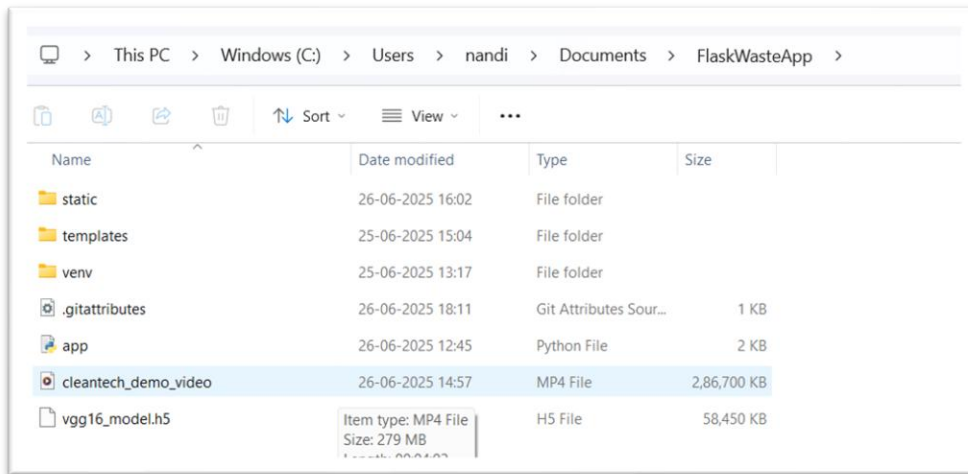
Building server-side script

Building HTML Pages:

For this project create three HTML files namely

- home.html
- result.html

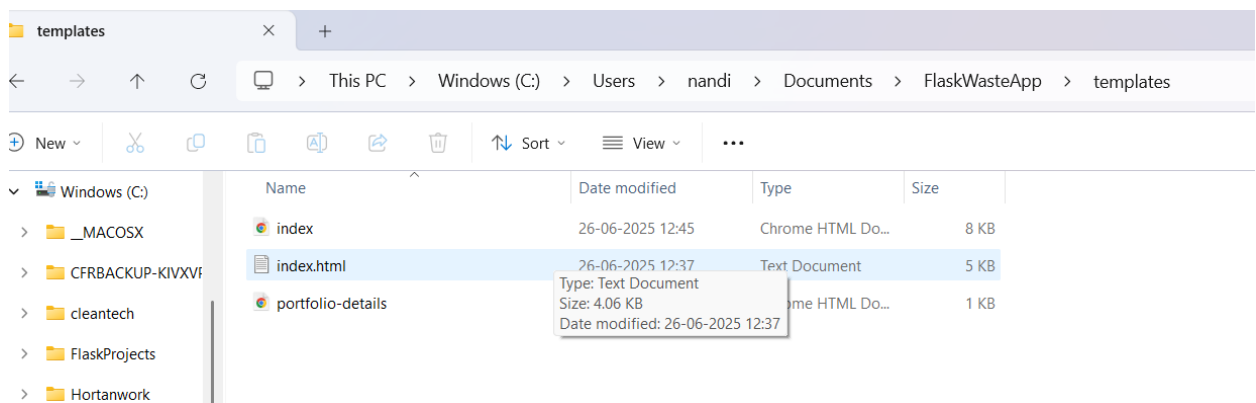
Building HTML Pages:



For this project create three HTML files namely

- index.html
- portofolio_details.html

And save them in the templates folder.



Build Python code:

The development of python code is in the form **app.py**

```
from flask import Flask, render_template, request, jsonify
import os
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask_cors import CORS

app = Flask(__name__)
CORS(app) # This enables CORS for all routes (important for frontend access)

model = load_model('vgg16_model.h5')
classes = ['Biodegradable', 'Recyclable', 'Trash']

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

@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part'})

    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No selected file'})

    # Save uploaded file
    filepath = os.path.join('static', file.filename)
    file.save(filepath)

    # Preprocess image
    img = image.load_img(filepath, target_size=(224, 224))
    img_tensor = image.img_to_array(img)
    img_tensor = np.expand_dims(img_tensor, axis=0)
    img_tensor /= 255.0

    # Predict
    prediction = model.predict(img_tensor)
    predicted_class = classes[np.argmax(prediction)]

    return jsonify({
        'prediction': predicted_class,
        'image_path': '/' + filepath
    })

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

Run the web application:

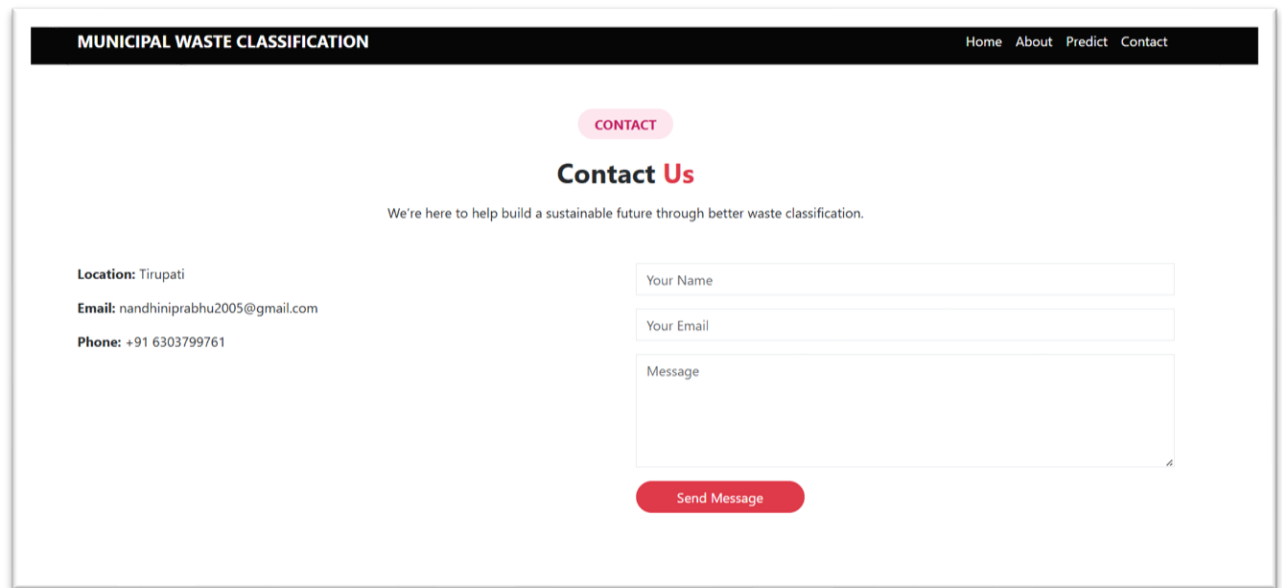
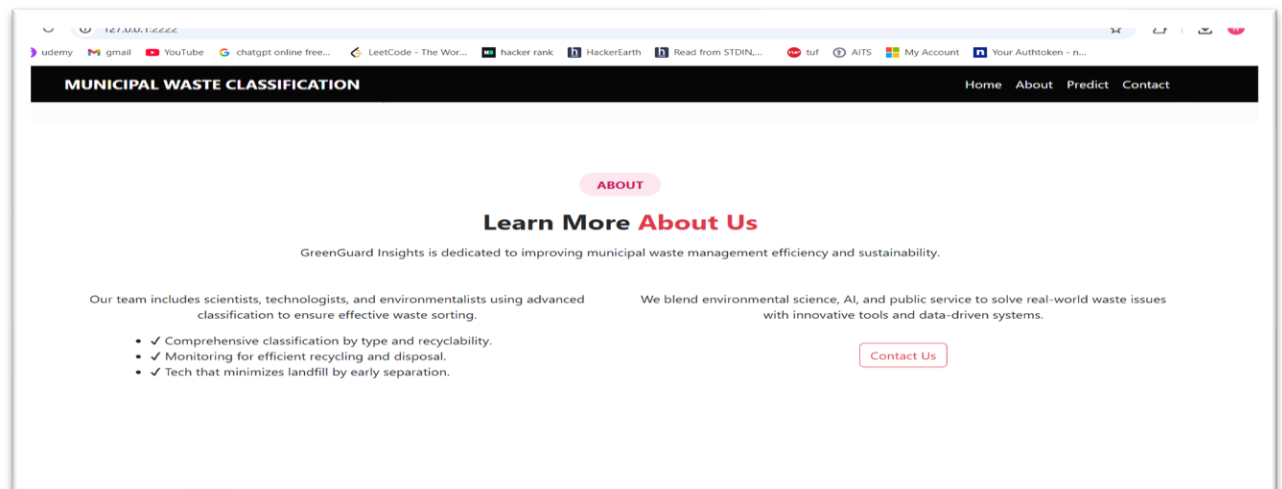
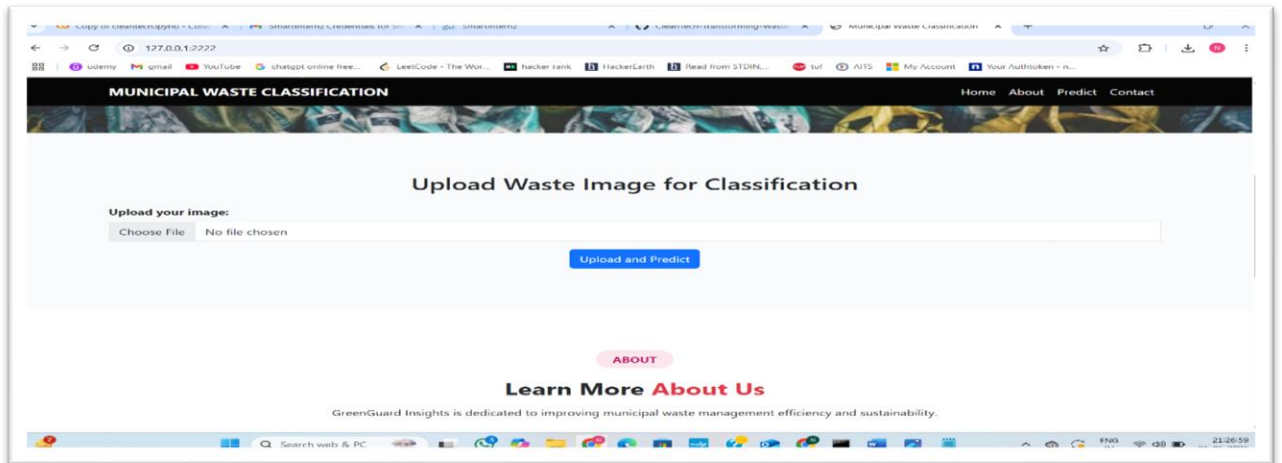
UI Image preview:

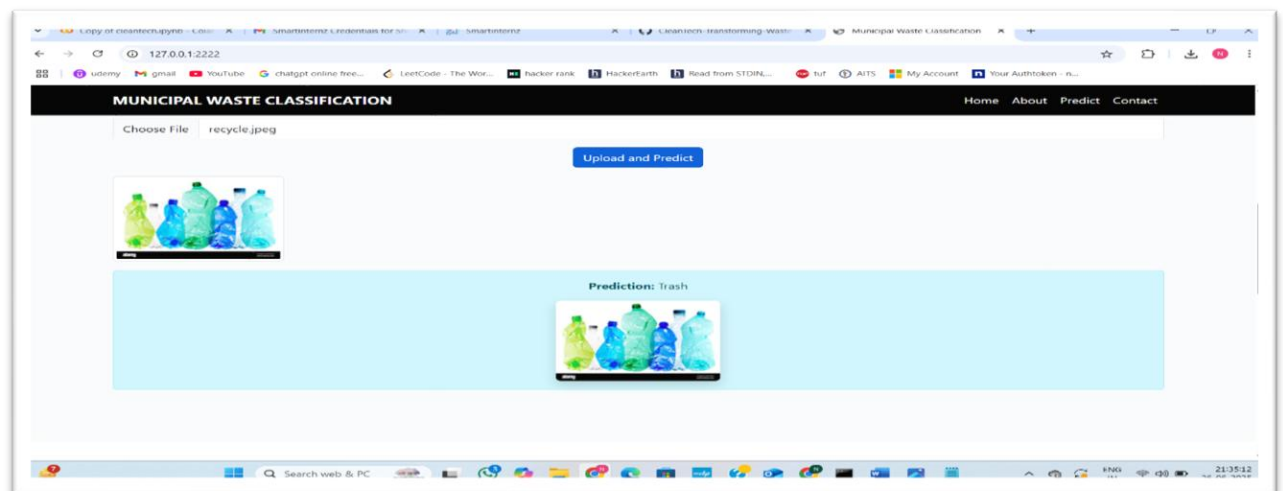
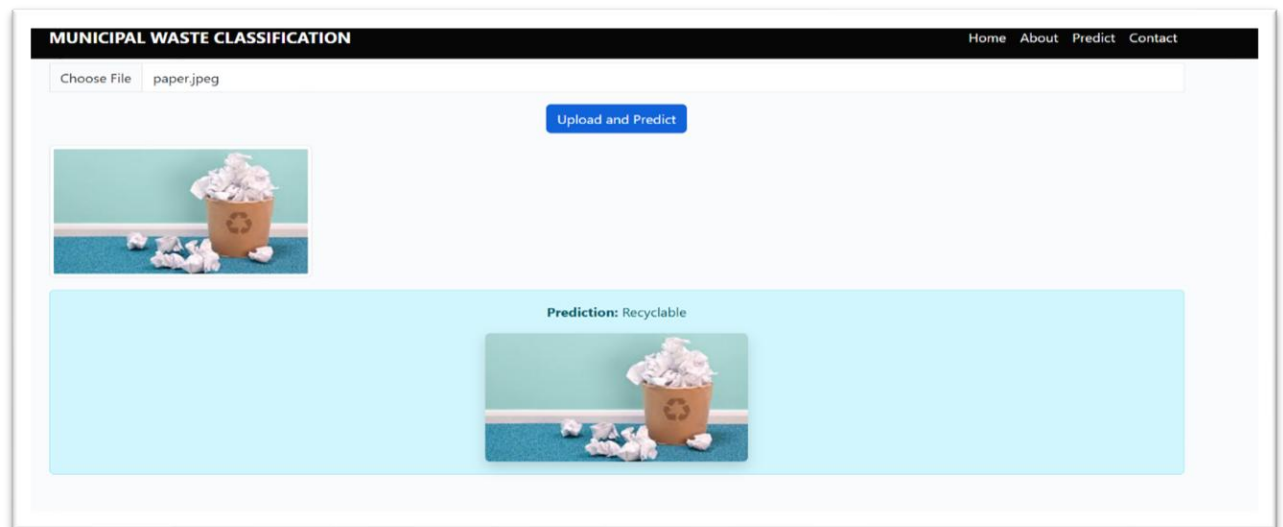
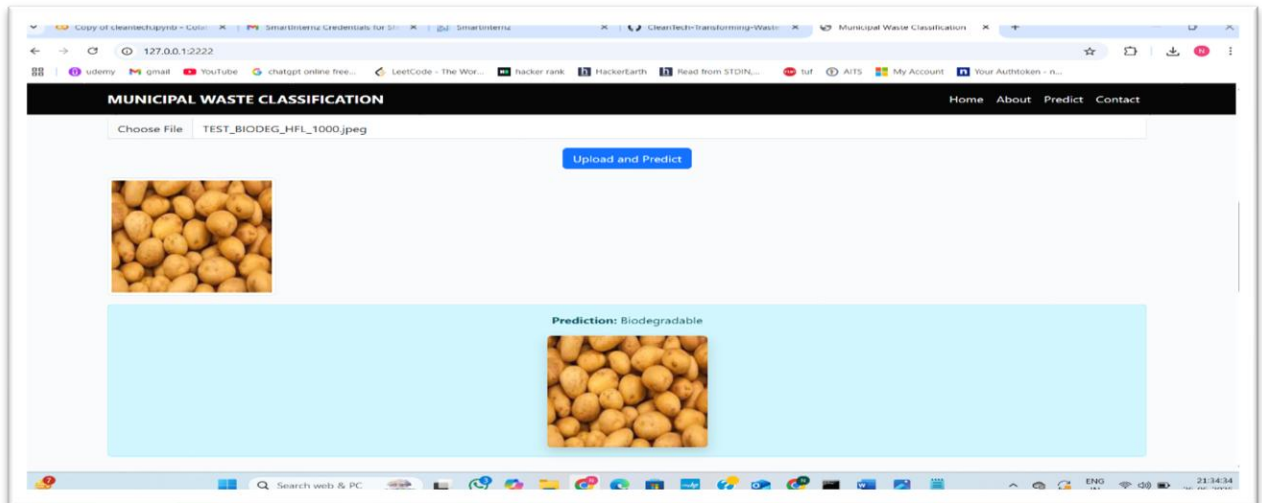
Let's see what our index.html page looks like:

```
Microsoft Windows [Version 10.0.26100.4202]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>cd C:\Users\nandi\Documents\FlaskWasteApp
C:\Users\nandi\Documents\FlaskWasteApp>venv\Scripts\activate
(venv) C:\Users\nandi\Documents\FlaskWasteApp>python app.py
2025-06-26 21:26:58.996898: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to float
orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-06-26 21:26:59.498768: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to float
orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2025-06-26 21:26:59.522888: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in perf
To enable the following instructions: SSE4.1 SSE4.2 AVX AVX2 AVX-VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
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 m
WARNING:tensorflow: > Debugger is active!
INFO:werkzeug: * Debugger PIN: 144-419-216
```







Run Your FlaskWasteApp – All-in-One CMD Guide

- `cd C:\Users\nandi\Documents\FlaskWasteApp`
- `venv\Scripts\activate`
- `pip install numpy pandas scikit-learn matplotlib seaborn tensorflow flask pillow`
- `python app.py`

Then open your browser and go to:

<http://127.0.0.1:2222>

-----*** THANKYOU ***-----