Ganthula Hyma
Email :
Ganthulahyma0123@Gmail.Com
BCA semester
Hall ticket no.:
23420161047008
College :
S.S.G.S Degree College

Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

Smart Sorting is an innovative project focused on enhancing the precision and efficiency of detecting rotten fruits and vegetables using cutting-edge transfer learning techniques. By leveraging pre-trained deep learning models and adapting them to specific datasets of fruits and vegetables, this project aims to revolutionize the process of sorting and quality control in the agricultural and food industry.

Scenario 1:

In a large food processing plant, workers manually sort through thousands of fruits and vegetables daily to separate the rotten ones from the fresh produce. This process is time-consuming, prone to human error, and labor-intensive. By implementing a smart sorting system that utilizes transfer learning for image recognition, the plant can automate this task. Cameras installed along the conveyor belts capture images of the produce. The system, trained on a vast dataset of images of both fresh and rotten produce, quickly and accurately identifies and sorts out the rotten items.

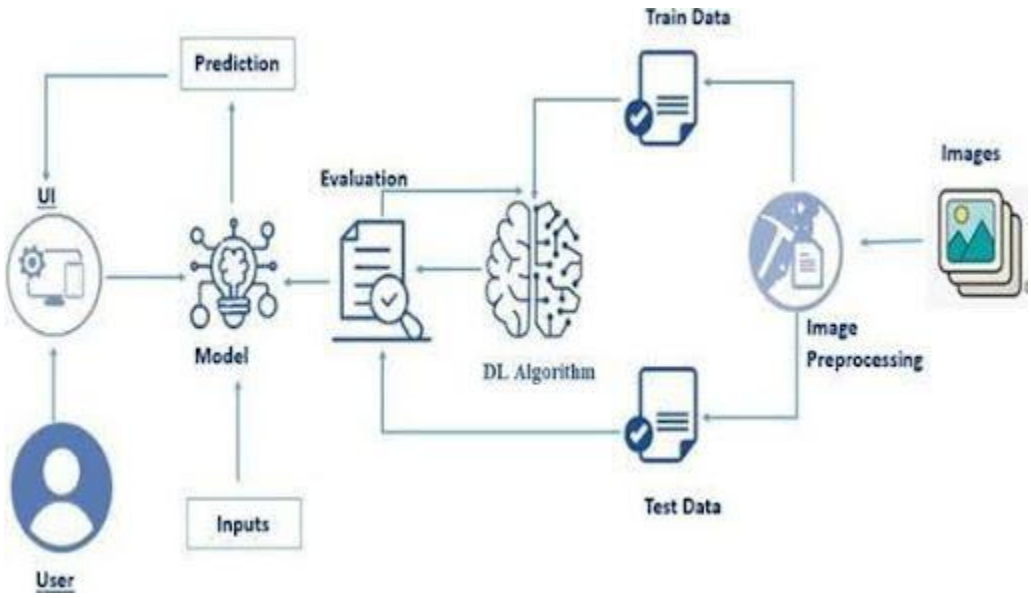
Scenario 2:

Supermarkets receive large shipments of fruits and vegetables, and ensuring the freshness of these products is crucial to maintaining customer satisfaction and reducing waste. A smart sorting system using transfer learning can be deployed at the receiving docks. As the shipments arrive, the system scans the produce in real-time, identifying any rotten items before they are stocked on the shelves. This ensures only fresh produce reaches the consumers, thereby enhancing the store's reputation for quality and freshness.

Scenario 3:

In modern smart homes, refrigerators equipped with smart sorting technology can help families reduce food waste. Using transfer learning algorithms, cameras inside the fridge continuously monitor the condition of stored fruits and vegetables. The system can alert users via a smartphone app when it detects any items starting to rot, suggesting that they should be consumed soon. This proactive approach helps households manage their food better, reducing waste and saving money.

Architecture:



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 e

Prior Knowledge

You must have prior knowledge of the following topics to complete this project.

DL Concepts

Neural Networks::
https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/

Deep Learning Frameworks::
https://www.knowledgehut.com/blog/data-science/pytorch-vs-tensorflow

Transfer Learning:
https://towardsdatascience.com/a-demonstration-of-transfer-learning-of-vgg-convolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a

VGG16: https://www.geeksforgeeks.org/vgg-16-cnn-model/

Convolutional Neural Networks (CNNs):
https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/
s://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning

Overfitting and Regularization:
https://www.analyticsvidhya.com/blog/2021/07/prevent-overfitting-using-regularization-techniques/

Optimizers:
https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/

Flask Basics: 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

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

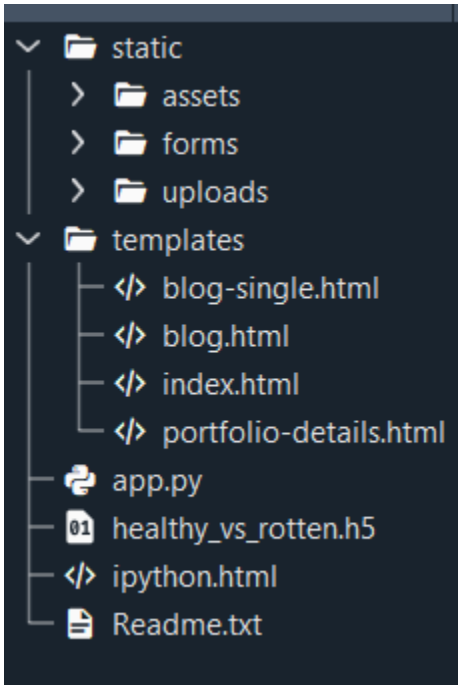
Import the model-building libraries
Initializing the model
Training and testing the model
Evaluating the performance of the model

Save the model

Application Building
Create an HTML file

Project Structure

Create the Project folder which contains files as shown below



We are building a Flask application with HTML pages stored in the templates folder and a Python script app.py for scripting.

Healthy_vs_rotten.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

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

Activity 1: Download the dataset

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

In this project, we have used 28 classes of fruits and vegetables 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 os
import shutil
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import shutil
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.applications.resnet50 import ResNet50
from tensorflow.keras.layers import Dense, Flatten
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
```

Import the necessary libraries as shown in the image.

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

```
import numpy as np
from sklearn.model_selection import train_test_split


# Set the path to the dataset
dataset_dir = '/content/Fruit And Vegetable Diseases 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)[:200]

    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.

```
# Specify the path to your image folder
folder_path = '/content/output_dataset/train/Apple__Healthy'  # 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 cucumber_rotten 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 cucumber_rotten.

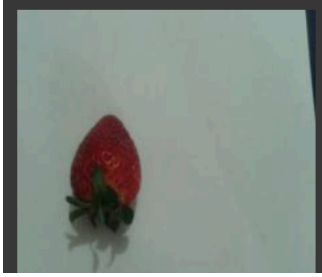
In the above code, I used class strawberry_rotten for prediction, This code randomly selects an image file from a specified folder (folder_path) containing JPEG, PNG, or JPEG files, and then displays the selected image using IPython's display function. It utilizes Python's OS and random modules for file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

```
# Specify the path to your image folder
folder_path = '/content/output_dataset/test/Strawberry__Healthy'  # 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 strawberry_healthy 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 strawberry_healthy .

```
# Specify the path to your image folder
folder_path = '/content/output_dataset/test/Cucumber__Rotten'  # 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 cucumber_rotten 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 cucumber_rotten.
In the above code, I used class strawberry_rotten 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 strawberry_rotten.

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.

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.

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.

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.

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.

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

```python
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
```

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

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

```
<keras.src.engine.input_layer.InputLayer object at 0x79c096fde230>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c096fde4d0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c0081b7a90>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bff7ef2f80>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c094405810>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c00834ba30>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bff6dad540>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c096fd2c20>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c094405360>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79c094405db0>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bfcc0fc490>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bff6dae7d0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bff6dad4b0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bff6dae020>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bfcc0fff10>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bfcc0fe0b0>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bfcc0fe770>
<keras.src.layers.convolutional.conv2d.Conv2D object at 0x79bfcc0fd300>
<keras.src.layers.pooling.max_pooling2d.MaxPooling2D object at 0x79bfcc0fe650>
```

```python
len(vgg.layers)
```

```
19
```

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

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

```python
output = Dense(28, activation ='softmax')(x)
```

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

```python
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

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080
```

```python
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 have 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=15,
                   steps_per_epoch=20,
                  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 1/15
20/20 [==============================] - 28s 1s/step - loss: 1.4851 - accuracy: 0.6100 - val_loss: 1.3616 - val_accuracy: 0.6390
Epoch 2/15
20/20 [==============================] - 25s 1s/step - loss: 1.2048 - accuracy: 0.6750 - val_loss: 1.4731 - val_accuracy: 0.6336
Epoch 3/15
20/20 [==============================] - 31s 2s/step - loss: 1.1071 - accuracy: 0.7075 - val_loss: 1.6028 - val_accuracy: 0.6023
Epoch 4/15
20/20 [==============================] - 25s 1s/step - loss: 0.8216 - accuracy: 0.7675 - val_loss: 1.1175 - val_accuracy: 0.6979
Epoch 5/15
20/20 [==============================] - 26s 1s/step - loss: 0.7029 - accuracy: 0.7875 - val_loss: 1.2511 - val_accuracy: 0.6836
Epoch 6/15
20/20 [==============================] - 26s 1s/step - loss: 0.8559 - accuracy: 0.7575 - val_loss: 1.2702 - val_accuracy: 0.6685
Epoch 7/15
20/20 [==============================] - 26s 1s/step - loss: 0.6845 - accuracy: 0.8100 - val_loss: 1.0867 - val_accuracy: 0.7265
Epoch 8/15
20/20 [==============================] - 25s 1s/step - loss: 0.5509 - accuracy: 0.8325 - val_loss: 1.2089 - val_accuracy: 0.7069
Epoch 9/15
20/20 [==============================] - 27s 1s/step - loss: 0.5796 - accuracy: 0.8400 - val_loss: 0.8013 - val_accuracy: 0.7802
Epoch 10/15
20/20 [==============================] - 25s 1s/step - loss: 0.4136 - accuracy: 0.8850 - val_loss: 0.9262 - val_accuracy: 0.7560
Epoch 11/15
20/20 [==============================] - 26s 1s/step - loss: 0.4959 - accuracy: 0.8575 - val_loss: 1.0332 - val_accuracy: 0.7292
Epoch 12/15
20/20 [==============================] - 27s 1s/step - loss: 0.5788 - accuracy: 0.8300 - val_loss: 0.8914 - val_accuracy: 0.7587
```

```python
vgg16.save('healthy_vs_rotten')
```

Testing Model & Data Prediction

Testing the model

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

```
labels=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27]
```

Testing class - 1

```
img_path = '/content/output_dataset/train/Bellpepper__Healthy/freshPepper (104).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 [==============================] - 0s 128ms/step
array([[0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

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

```
4
```

Testing class-2

```
img_path = '/content/output_dataset/train/Mango__Rotten/153.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 [==============================] - 0s 19ms/step
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

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

```
17
```

Testing class-3

```
[75] img_path ='/content/output_dataset/train/Orange__Healthy/Screen Shot 2018-06-12 at 11.55.05 PM.png'
```

```
[76] 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 [==============================] - 0s 21ms/step
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

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

```
12
```

Testing class-4

```
[79] img_path ='/content/output_dataset/train/Cucumber__Healthy/freshCucumber (127).jpg'
```

```
[80] 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 [==============================] - 0s 20ms/step
array([[0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

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

Testing class-5

```
[82] img_path ='/content/output_dataset/train/Potato__Rotten/rottenPotato (113).jpg'
```

```
[83] 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 [==============================] - 0s 21ms/step
array([[0.000000e+00, 0.000000e+00, 0.000000e+00, 5.546745e-37,
        0.000000e+00, 0.000000e+00, 0.000000e+00, 0.000000e+00,
        0.000000e+00, 0.000000e+00, 0.000000e+00, 0.000000e+00,
        0.000000e+00, 0.000000e+00, 0.000000e+00, 0.000000e+00,
        0.000000e+00, 0.000000e+00, 0.000000e+00, 0.000000e+00,
        0.000000e+00, 0.000000e+00, 0.000000e+00, 1.000000e+00,
        0.000000e+00, 0.000000e+00, 0.000000e+00, 0.000000e+00]],
      dtype=float32)
```

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

```
23
```

Saving the model

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

```
vgg16.save('healthy_vs_rotten')
```

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:

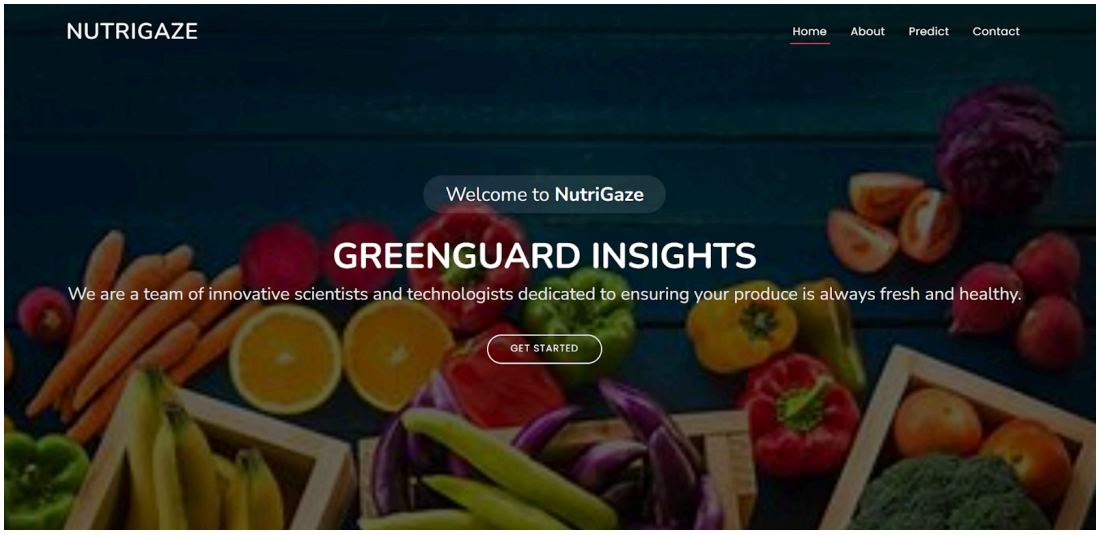
For this project create three HTML files namely

index.html

And save them in the templates folder.

UI Image preview:

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



ABOUT

Learn More About Us

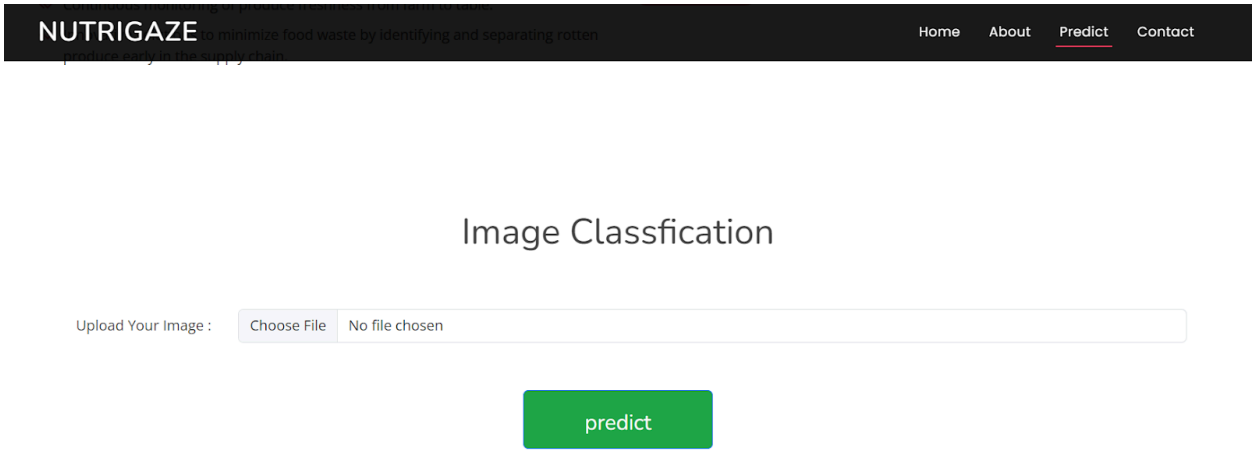
NutriGaze is a pioneering organization dedicated to enhancing the quality and safety of your fruits and vegetables.

Our team comprises innovative scientists, technologists, and agricultural experts committed to leveraging advanced detection technologies to ensure the produce you consume is fresh, healthy, and nutritious.

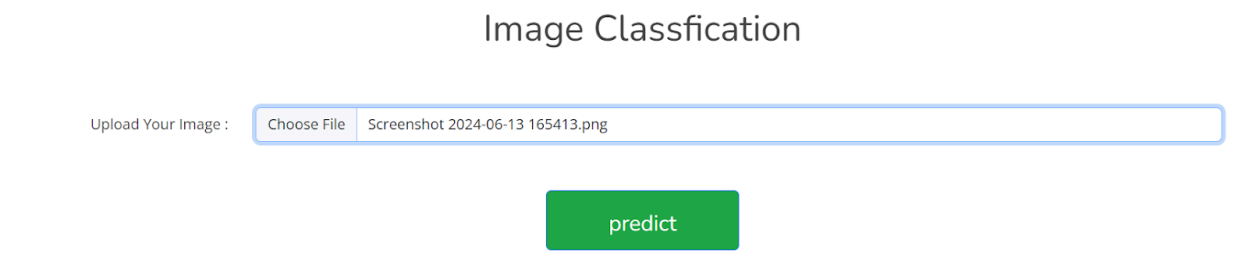
- Comprehensive analysis and grading of fruits and vegetables based on ripeness and nutritional content.
- Continuous monitoring of produce freshness from farm to table.
- Innovative solutions to minimize food waste by identifying and separating rotten produce early in the supply chain.

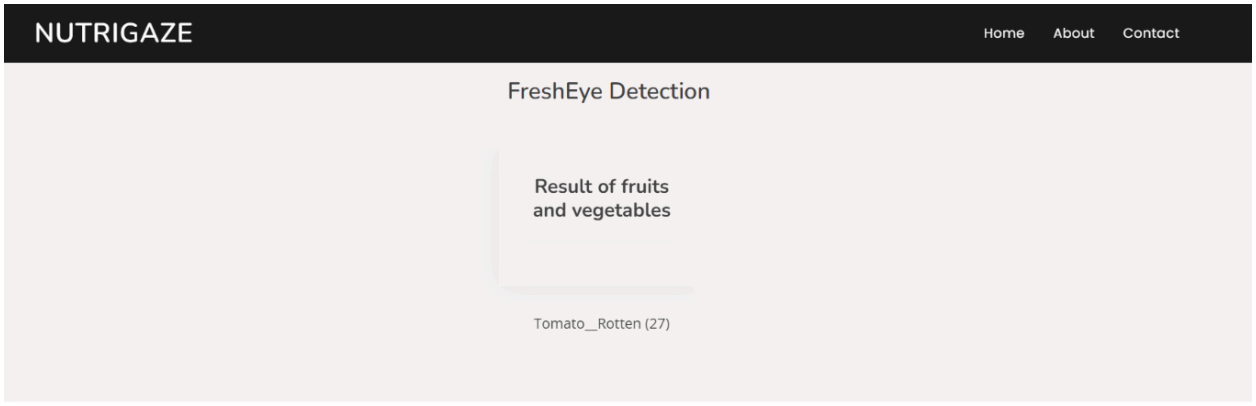
Our team is our greatest asset. We are a diverse group of experts in fields such as agricultural science, data analytics, software engineering, and food technology. Together, we bring a wealth of knowledge and experience to tackle the challenges of food quality and safety.

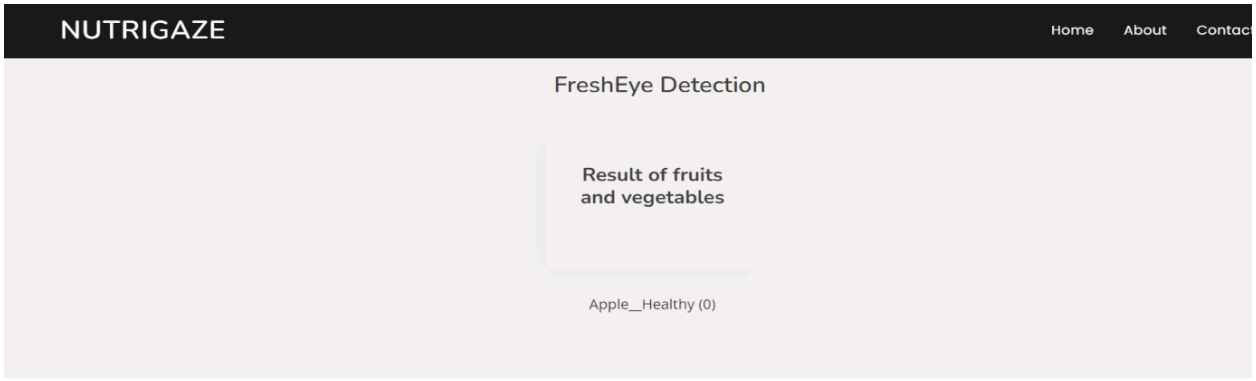
Learn More



Now when you click on the inspect button further in the top right corner you will get redirected to Inspect.html







Build Python code:
Import the libraries

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.

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:

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.

Build Python code:
Import the libraries

```
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('healthy_vs_rotten.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=['Apple__Healthy (0)','Apple__Rotten (1)','Banana__Healthy (2)',
               'Banana__Rotten (3)','Bellpepper__Healthy (4)','Bellpepper__Rotten (5)',
               'Carrot__Healthy (6)','Carrot__Rotten (7)', 'Cucumber__Healthy (8)','Cucumber__Rotten (9)','Grape__Healthy (10)',
               'Grape__Rotten (11)','Guava__Healthy (12)','Guava__Rotten (13)',
               'Jujube__Healthy (14)',
               'Jujube__Rotten (15)', 'Mango__Healthy (16)', 'Mango__Rotten (17)', 'Orange__Healthy (18)',
               'Orange__Rotten (19)', 'Pomegranate__Healthy (20)', 'Pomegranate__Rotten (21)', 'Potato__Healthy (22)',
               'Potato__Rotten (23)', 'Strawberry__Healthy (24)', 'Strawberry__Rotten (25)', 'Tomato__Healthy (26)', 'Tomato__Rotten (27)'
              ]
        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:

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

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

Run the web 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/flask2/app.py', wdir='C:/
Users/santu/Downloads/flask2')
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
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)
```