

Transfer Learning for Identifying Rotten Fruits and Vegetables

Team ID: LTVIP2026TMIDS463

Team Size: 3

Team Leader: Burepalli varshitha

Team member: Dirisanapu Vanaja

Team member: Jeetham Uday Kiran

INTRODUCTION:

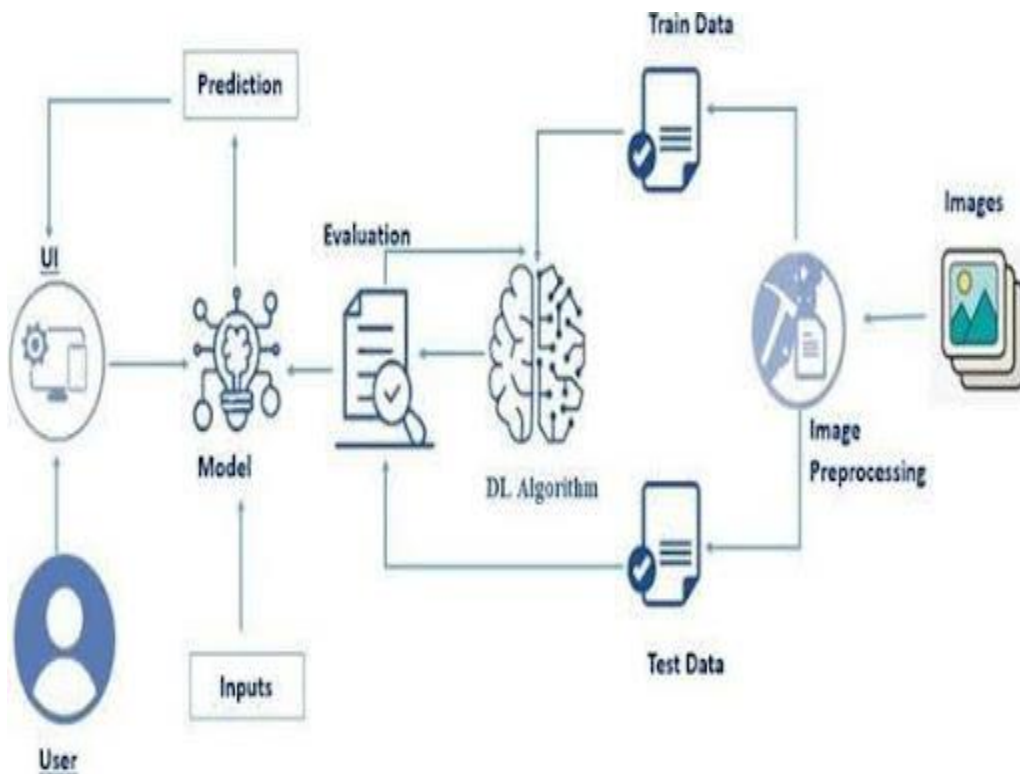
- Fruits and vegetables are essential food items but they spoil easily due to environmental condition and improper storage.
- Manual inspection of rotten fruits and vegetable is time-consuming costly, and may lead to human errors.
- Food wastage is a major problem in agriculture, supply chains, and markets due to the lack of automated quality checking systems.
- Machine Learning and Deep Learning techniques help in analysing images to automatically detect spoilage.
- Image classification using convolutional Neural Networks (CNNs) is widely used for identifying visual defects such as colour change, models, and texture damage.
- Training deep learning models from scratch requires large datasets and high computational power.
- Transfer Learning overcomes this limitation by using pre-trained models like VGG16, Res Net, Mobile Net, and inception.
- These pre-trained models are already trained on large dataset (Image Net) and can extract meaningful image feature.
- By fine-tuning these models, fruits and vegetables can be classified as fresh or rotten with high accuracy.
- This approach reduces training time, improves performance, and works well even with small datasets.
- The system can be used in agriculture markets, supermarkets, food processing industries, and smart framing applications.
- Overall, transfer learning provides an efficient and reliable solution for automatic detection of rotten fruits and vegetables.

DETECTION:

Detection of rotten fruits and vegetables involves automatically identifying whether a given fruit or vegetable is fresh or spoiled based on its visual features. In this approach, images are collected and preprocessed to improve quality through resizing and normalization. Transfer learning is used by employing pre-trained convolutional neural network models such as VGG16, Res Net, or Mobile Net, which are already trained on large datasets like ImageNet. These models extract important features such as colour changes, texture irregularities, dark spots, and mold growth that indicate spoilage. The final layers of the network are fine-tuned to classify the input image as fresh or rotten. This automated detection method provides accurate and consistent results, reduces manual inspection effort, and helps minimize food wastage in agricultural and food supply chain systems.

During detection, the trained model processes a new image, extracts features, and predicts the category along with a confidence score. The performance of the detection system is evaluated using metrics like accuracy, precision, recall, F1-score, and confusion matrix. This automated approach reduces dependency on manual inspection, increases speed and consistency, and helps reduce food wastage in storage units, supermarkets, food industries, and agricultural markets. Furthermore, it can be integrated into IoT-based smart farming systems, mobile applications, and robotic sorting machines for real-time monitoring and better supply chain management.

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 tensorflow” and click enter.
 - Type “pip install Flask” and click enter.

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=Ij4l_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:

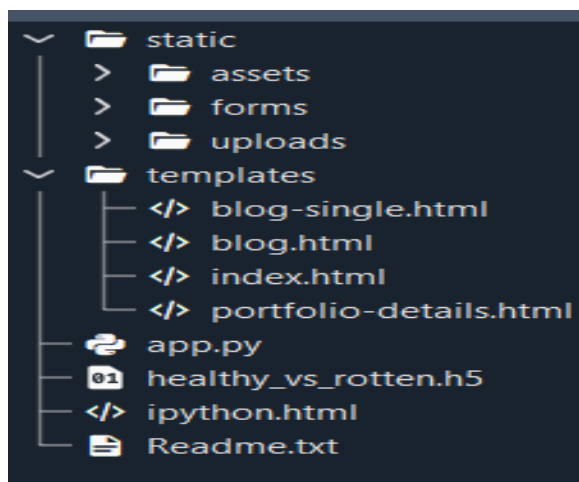
- The user interacts with the UI (User Interface) to choose the image.
- The chosen image is analyzed by the model which is integrated with the flask application.
- Once the model analyses the input the prediction is showcased on the UI

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
 - Build python code

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 data base

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 the necessary libraries as shown in the image.

```

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

```

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.

```

!mkdir ~/.kaggle

!cp kaggle.json ~/.kaggle

!kaggle datasets download -d muhammad0subhan/fruit-and-vegetable-disease-healthy-vs-rotten

Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Dataset URL: https://www.kaggle.com/datasets/muhammad0subhan/fruit-and-vegetable-disease-healthy-vs-rotten
License(s): CC0-1.0
Downloading fruit-and-vegetable-disease-healthy-vs-rotten.zip to /content
100% 4.76G/4.77G [00:41<00:00, 136MB/s]
100% 4.77G/4.77G [00:41<00:00, 124MB/s]

!unzip /content/fruit-and-vegetable-disease-healthy-vs-rotten.zip

```

```

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.")

```

```

# 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'
)

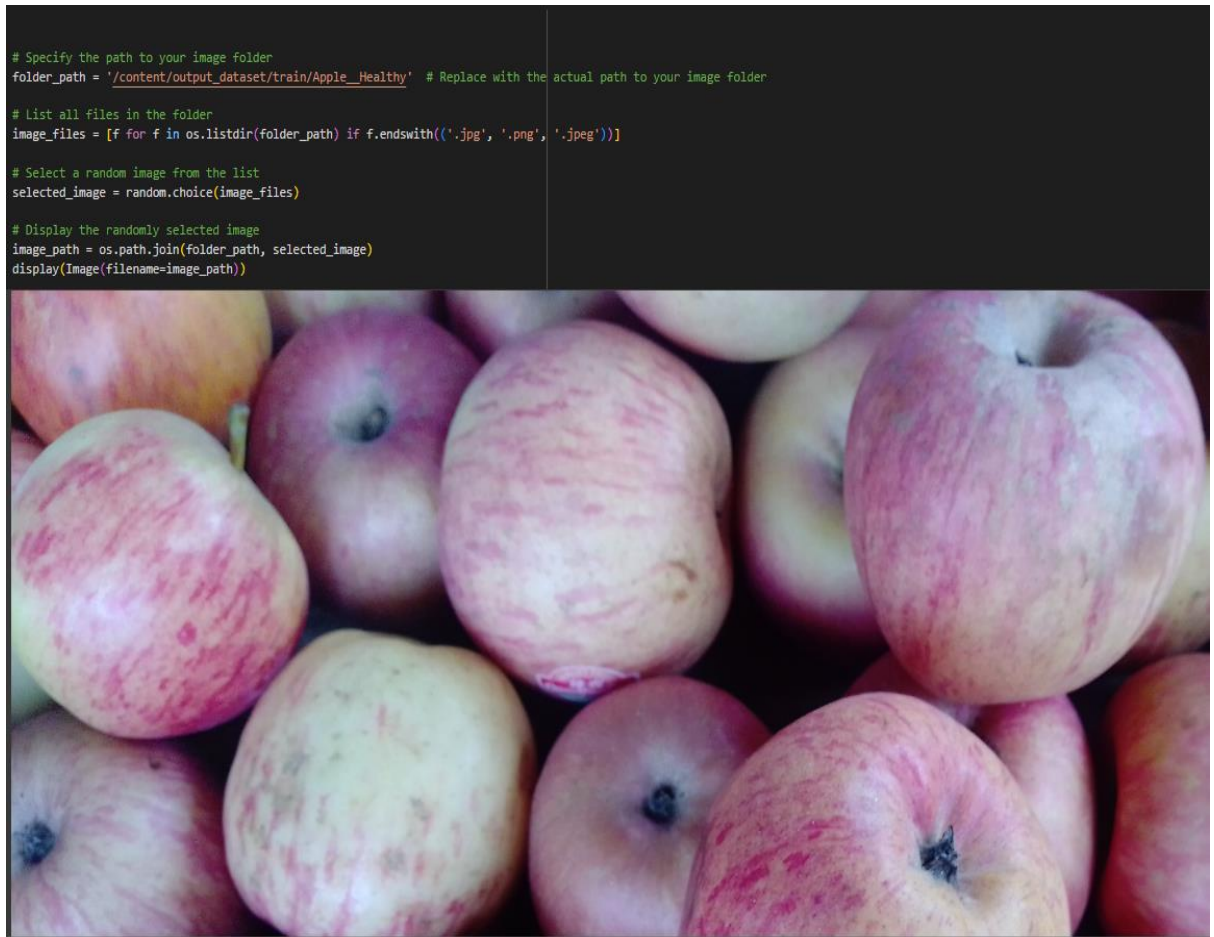
test_generator = val_test_datagen.flow_from_directory(
    test_dir,
    target_size=IMG_SIZE,
    batch_size=32,
    class_mode='binary',
    shuffle=False # Do not shuffle test data
)

# 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 I Python'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.



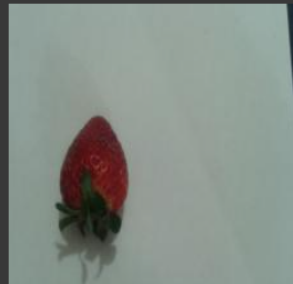
In the above code, I used class Apple_ healthy 0 for prediction, This code randomly selects an image file from a specified folder (folder_ path) containing JPEG, PNG, or JPEG files, and then displays the selected image using I Python's display function. It utilizes Python's OS and random modules for file manipulation and random selection, respectively. And It has predicted correctly as Apple_ healthy 0.

```
# 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 I Python'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))
```



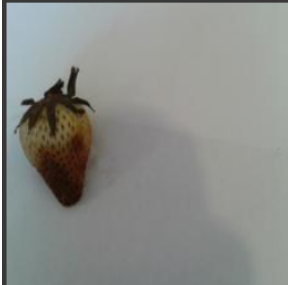
In the above code, I used class cucumber_ rotten for prediction ,This code randomly selects an image file from a specified folder (folder_ path) containing JPEG, PNG, or JPEG files, and then displays the selected image using I Python's display function. It utilizes Python's OS and random modules for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

```
# Specify the path to your image folder
folder_path = '/content/output_dataset/test/Strawberry_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 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 Python'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 labelled 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) , #5 , 15 , 32, 50

Found 3358 images belonging to 28 classes.
Found 1120 images belonging to 28 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.

```

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

```

```

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

```

```

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>

```

```

len(vgg.layers)

```

```

19

```

```

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

```

```

x= Flatten()(vgg.output)

```

```

output = Dense(28, 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
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

```

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

```

```
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., 0.]], dtype=float32)
```

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

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.,
        0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]], dtype=float32)
```

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

17

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:

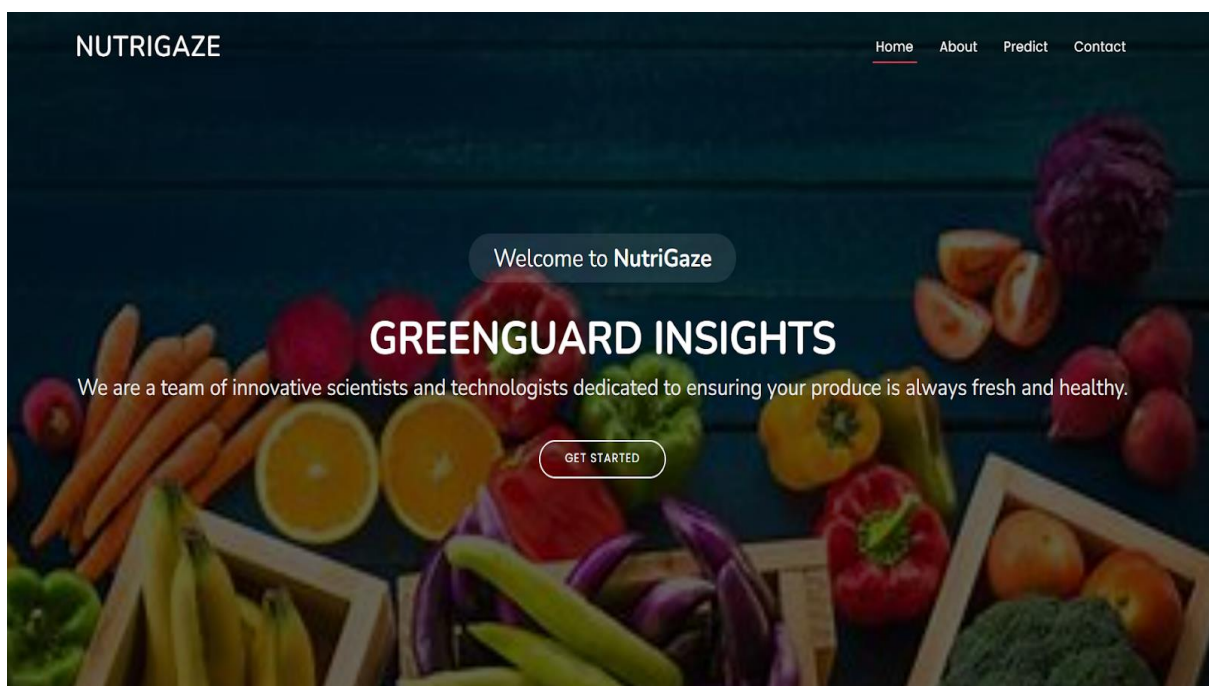
For this project create three HTML files namely

- 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](#)

Image Classification

Upload Your Image :

Choose File

No file chosen

predict

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

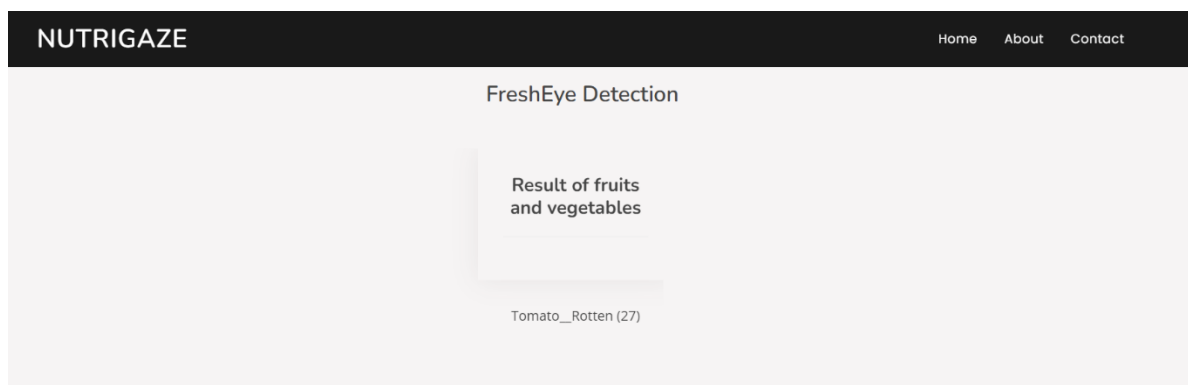
Let's look at what our inner.html file looks like and test the model:

Test Class tomato rotten (27):

Image Classification

Upload Your Image : Screenshot 2024-06-13 165413.png

Now when you click on the predict button you will get output down to the image itself
Let's look at what our output looks like



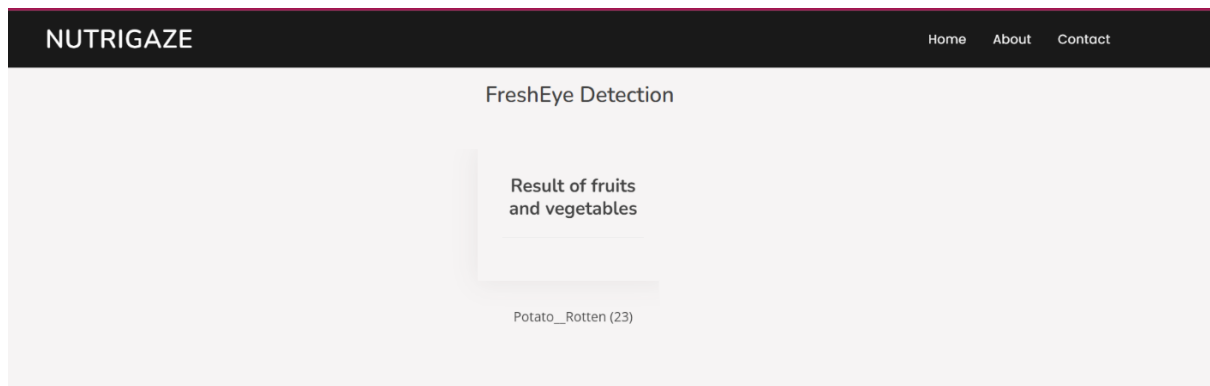
Test Class potato rotten (23): Now

Image Classification

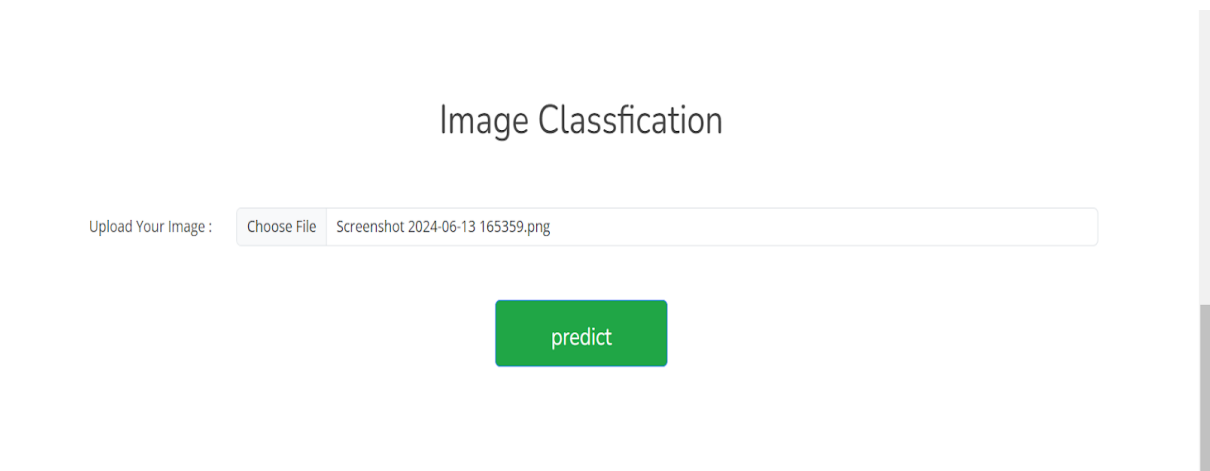
Upload Your Image : Screenshot 2024-06-13 165313.png

Now when you click on the predict button you will get output down to the image itself

Let's look how our output looks like:



Test Class tomato healthy (26):



Now when you click on predict button you will get output down to the image itself

Let's look how our output looks like:

FreshEye Detection

Result of fruits
and vegetables

Tomato_Healthy (26)

Test Class strawberry_ healthy (24):

Image Classification

Upload Your Image :

Choose File

Screenshot 2024-06-13 164630.png

predict

Now when you click on predict button you will get output down to the image itself

Let's look how our output looks like:

FreshEye Detection

Result of fruits
and vegetables

Strawberry__Healthy (24)

Test Class Apple_ healthy (0):

Image Classification

Upload Your Image :

Choose File

Screenshot 2024-06-13 164625.png

predict

Now when you click on predict button you will get output down to the image itself

Let's look how our output looks like:

FreshEye Detection

Result of fruits
and vegetables

Apple__Healthy (0)

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 be 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

- 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)
```

CONCLUSION:

In this project, transfer learning was successfully applied to detect rotten fruits and vegetables using image classification techniques. By using pre-trained convolutional neural network models such as VGG16, Res Net, Mobile Net, and Inception, the system was able to identify visual features like colour changes, module, spots, and texture variations that indicate spoilage. Transfer learning reduced the need for a large dataset and minimized training time while still achieving high accuracy