Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

MENTOR NAME – I SUNEETHA

TEAM ID - LTVIP2025TMID35577

TEAM LEADER

NAME – T SRUTHI

MAIL - thinnellurusruthi@gmail.com

IDNO:23AK5A0422

TEAM MEMBERS

NAME – P VARSHITHA

MAIL - varshithapoluru@gmail.com

IDNO: 23AK5A0425

NAME – K YASHWITHA REDDY

MAIL - yaswithareddy028@gmail.com

IDNO: 23AK5A0428

NAME – V YESWANTHI

MAIL - yeswanthi082@gmail.com

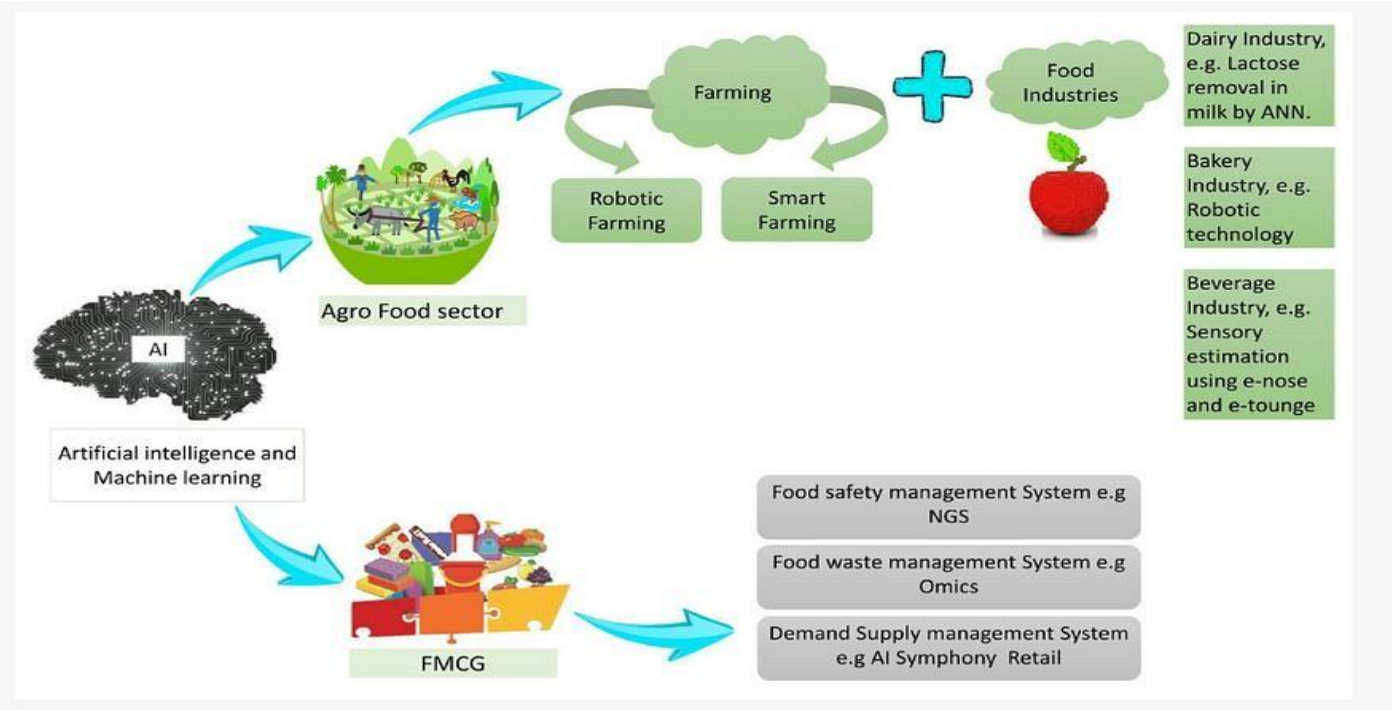
IDNO: 23AK5A0429

INTRODUCTION:

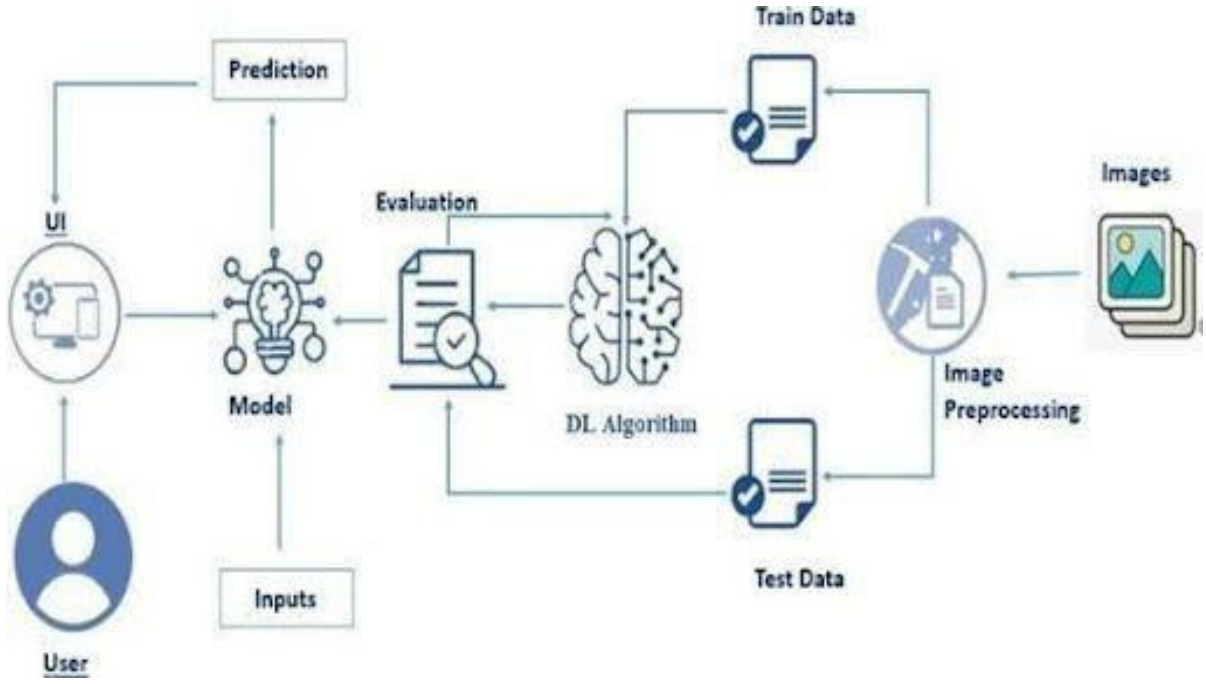
In Today's Agricultural And Food Industries, Ensuring The Quality Of Fruits And Vegetables Is Crucial To Minimize Waste, Maintain Hygiene, And Deliver Safe, Fresh Produce To Consumers. Manual Sorting Is Time-Consuming, Prone To Human Error, And Inefficient On A Large Scale. The Smart Sorting Project Addresses This Problem By Using Transfer Learning, A Powerful Machine Learning Technique, To Automatically Identify And Classify Rotten Versus Fresh Fruits And Vegetables Through Images.

By Leveraging Pre-Trained Deep Learning Models Such As Mobilenet, Resnet, Or VGG, The Project Adapts These Networks To Recognize Spoilage Patterns In Produce With Minimal Training Data. This Approach Reduces The Computational Cost And Speeds Up The Development Process While Delivering High Accuracy.

The System Can Be Deployed Through A Web Or Mobile Interface Where Users Can Upload Images Of Produce, And The Model Instantly Classifies Them As Fresh Or Rotten—Helping Farmers, Retailers, And Supply Chain Operators Optimize Sorting And Reduce Food Waste.



Architecture:



The diagram illustrates the architecture of a Deep Learning system for image analysis, specifically for tasks like early detection of Diabetic Retinopathy based on the context of the search results.

Explanation of the Diagram:

- **User and UI:**

The process begins with a "User" interacting with a "UI" (User Interface) to provide "Inputs".

- **Model and Prediction:**

These inputs are fed into a "Model," which uses a "DL Algorithm" (Deep Learning Algorithm) to perform "Prediction".

- **Image Preprocessing and Data:**

"Images" are subjected to "Image Preprocessing" before being split into "Train Data" and "Test Data." This pre processed data is crucial for training and evaluating the Deep Learning Algorithm.

- **DL Algorithm and Evaluation:**

The "DL Algorithm" is trained using the "Train Data" and its performance is assessed using the "Test Data" through an "Evaluation" process. This evaluation helps refine the "Model".

Prerequisites:

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 Frame works:: https://www.knowledgehut.com/blog/data-science/pytorch-vstensorflow**
- **Transfer Learning: https://towardsdatascience.com/a-demonstration-of-transfer-learning-of-vggconvolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a**
- **VGG16: https://www.geeksforgeeks.org/vgg-16-cnn-model/**
- **ConvolutionalNeuralNetworks (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/preventoverfitting-using-regularization-techniques/**
- **Optimizers: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deeplearning-optimizers/**
- **Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0**

Project Objectives :

1. Deep Learning (DL) :

- Use **Convolutional Neural Networks (CNNs)** for feature extraction from images.
- Train models to detect spoilage based on visual features.
- Reduce manual sorting effort and human error.
- Improve accuracy and speed of the sorting process.
- Support real-time decision-making for quality control.
- Contribute to reducing food waste through efficient sorting.

2. Convolutional Neural Networks (CNNs) :

- A powerful type of neural network used for **image processing**.
- CNNs automatically learn features like **color, shape, and texture**.
- **Key layers:**
- **Convolutional Layer** – Detects features in the image.
- **Pooling Layer** – Reduces image size while keeping important information.
- **Fully Connected Layer** – Makes the final classification decision.

3. Image Classification :

- The goal is to **classify** images into categories like:
- Fresh Apple / Rotten Apple
- Fresh Tomato / Rotten Tomato
- Requires a **labeled dataset of fruit and vegetable images.**

4. Transfer Learning :

- Instead of training a model from scratch (which needs lots of data), use a **pre-trained model**. •

  Examples: **ResNet, MobileNet, VGG16**
- Only the final layers are retrained for the new task (fruit sorting).

5. Data Collection and Preprocessing :

- Collect images of fruits and vegetables in **different conditions** (fresh and rotten).
- Resize images (e.g., 224x224), normalize pixels, and augment with:
- **Rotation**, **flip**, **zoom**, **brightness change** to improve model robustness.

6. Model Training :

- Dataset is split into:
- **Training set** (to learn),
- **Validation set** (to tune),
- **Test set** (to evaluate).
- Optimizer: **Adam**

- Loss function: **Categorical Crossentropy** • Evaluation metrics: **Accuracy**, **Precision**, **Recall**

7. Evaluation & Tuning :

- Use a **confusion matrix** to analyze correct and incorrect predictions.
- Tune hyperparameters (like learning rate or batch size) to improve performance.

8. Model Deployment :

- Save the model and integrate it into a **web or mobile app**.
- Users upload an image → model classifies it → result is displayed.

These are the **core ideas and techniques** behind building a smart sorting system using deep Learning.

Gain a broad understanding of data:

1. Types of Data Used :

- **ImageData:**

  High-quality images of fruits and vegetables from various angles, lighting conditions, and states (fresh, partially rotten, fully rotten).

- **LabelledData:**

  Each image is labeled as "fresh", "rotten", or "partially rotten". This is crucial for training the model.

- **Metadata(optional):**

  May include timestamp, location, or environmental conditions like humidity and temperature (useful for future enhancements)

2. Data Collection Sources :

- **PublicDatasets**:

  e.g., Fruits 360, Kaggle fruit classification datasets.

- **CustomImageCapture:**

  Using mobile cameras or Raspberry Pi + camera setup in a lab/farm/warehouse setting.

- **AugmentedData**:

  Rotated, flipped, zoomed versions of original images to increase data size artificially.

3. Data Preprocessing :

- **Resizing:**

  Standard image dimensions (e.g., 224x224) for deep learning models.

- **Normalization:**

  Scale pixel values between 0 and 1.

- **LabelEncoding:**

  Convert class labels to numbers (e.g., 0: fresh, 1: rotten).

- **DataSplitting:**

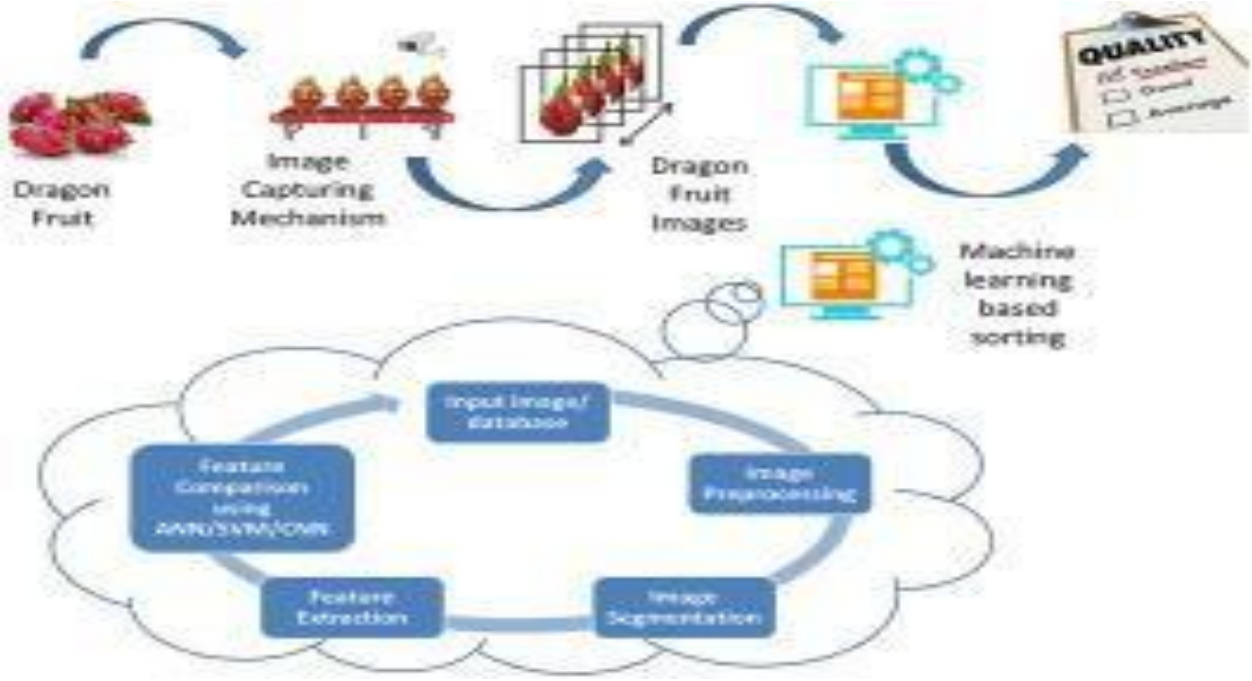
  Train, Validation, Test sets (commonly 70:20:10 or 80:10:10).

4. Role of Data in Smart Sorting

- **Trains the model to recognize patterns (color, texture, spots).**
- **Helps Transfer Learning models like ResNet, MobileNet, or VGG adapt faster with fewer images.**
- **Supports real-time predictions in web/mobile-based fruit sorting systems.**

5. Feedback Loop

- **After deployment, incorrect predictions can be collected and added back to the dataset for continuous learning and model improvement.**



In Smart Sorting, data is the foundation. High-quality, well-labeled images with balanced categories and proper preprocessing enable effective training using deep learning and transfer learning techniques,The better the data, the and more accurate your fruit and vegetable sortig system smarter

knowledge of pre-processing the data/transformation techniques on outliers and some visualization concept:

1. Data Pre-processing for Smart Sorting:

Pre-processing is the first step to prepare raw data (images of fruits/vegetables) for machine learning models.

Common pre-processing steps include:

- **Resizing images** to a standard shape (e.g., 224x224 pixels for transfer learning).
- **Normalizing pixel values** (scaling values from 0–255 to 0–1).
- **Removing noise** or poor-quality images.
- **Augmentation**: Creating new images by flipping, rotating, or zooming to increase data variety.

2. Handling Outliers:

Outliers are unusual data points (like very blurred or wrongly labeled images) that can reduce model Accuracy.

Techniques to handle them:

- **Visual Inspection**: Manually checking and removing clearly wrong data.
- **Statistical Methods**: Use tools like boxplots to find data points with extreme values.
- **Z-Score or IQR Method** (for numeric features if available): Identifies values far from the mean or median.

3. Transformation Techniques:

Transformation makes the data better suited for learning by:

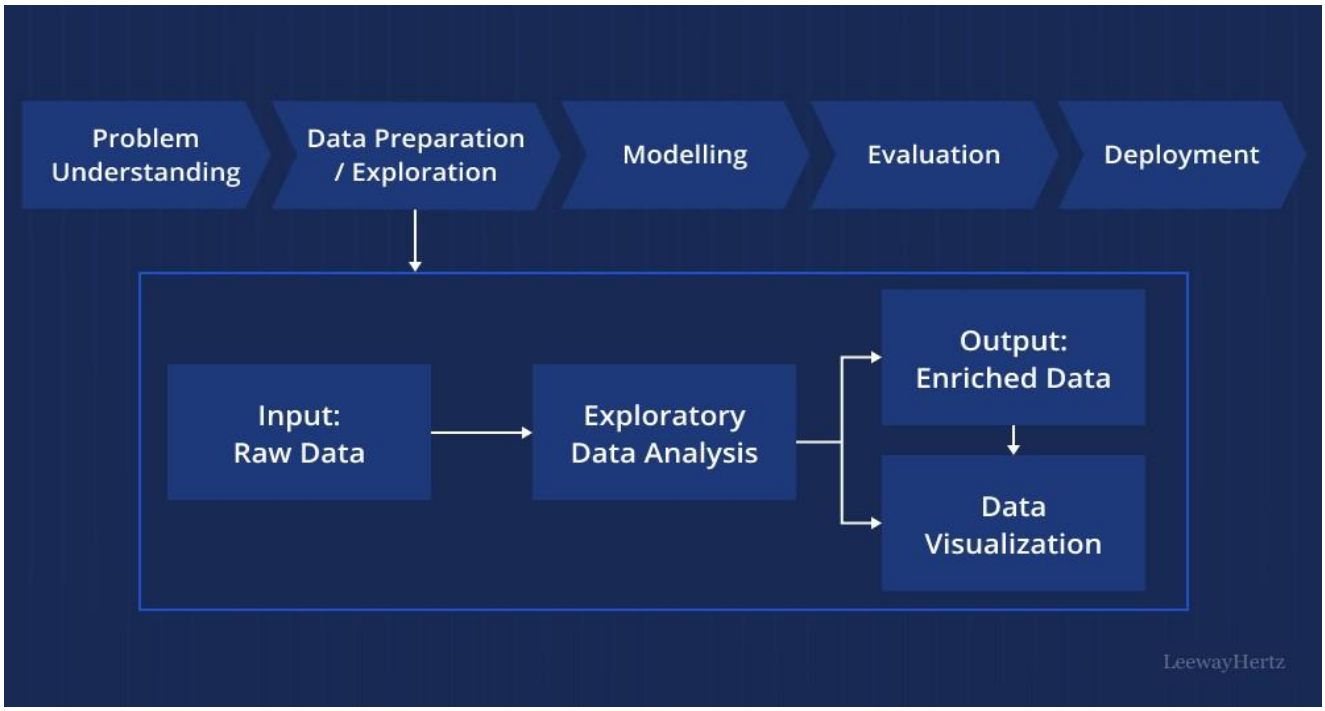
- **Scaling**: Standardizing image data (mean = 0, std = 1) helps in faster training.
- **Encoding Labels**: Fresh = 0, Rotten = 1 (convert labels to numbers).
- **Data Augmentation** (again): It acts like a transformation and helps balance the dataset

4. Visualization Concepts:

Visual tools help understand data quality and training progress.

Useful visualizations:

- **Class Distribution Chart**: Bar graph showing count of fresh vs. rotten images.
- **Sample Image Plots**: To check if images are clear and labeled properly.
- **Training Accuracy/Loss Graphs**: To monitor model learning over time.
- **Confusion Matrix**: Shows how well the model predicts each class.
- **t-SNE/ PCA plots** (advanced): Visualize image feature clusters after training.



Project Flow :

- **User Interaction & Prediction Display**:

  The user selects an image via a UI, which is then analyzed by a Flask-integrated model, with predictions showcased back on the UI.

- **Data Management**:

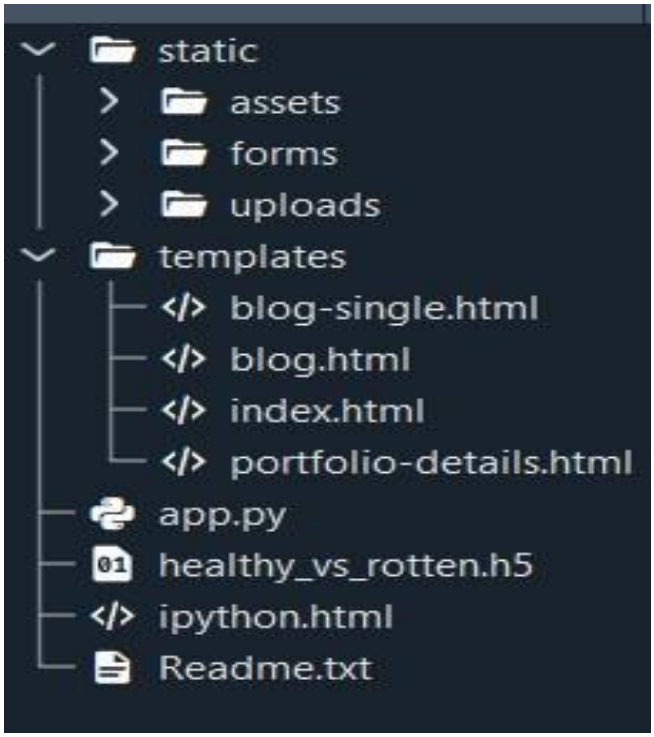
  This involves collecting or downloading the dataset, followed by pre-processing steps like data augmentation and splitting data for training and testing.

- **Model Development**:

This phase includes importing libraries, initializing, training, testing, and evaluating the model's performance, culminating in saving the trained model.

- **Application Development:**
  The final stage focuses on building the application by creating necessary HTML files and developing the Python code for functionality.

Project Structure :



- 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

Collect the dataset:

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

There are many popular open sources for collecting the data.

In this project, we have used 28 classes of fruits and vegetables data.

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.

There are several techniques for understanding the data. But here we have used some of it.

```
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
```

Read the Data:

- 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
```

```python
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 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.

```python
# 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))
```

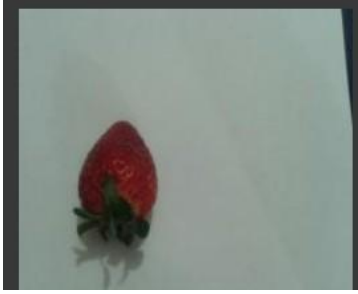
In the above code, I used class Apple_healthy 0 for prediction, This code randomly selects an image file from a specified folder (folder_path) containing JPEG, PNG, or JPEG files, and then displays the selected image using IPython's display function. It utilizes Python's OS and random modules for file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

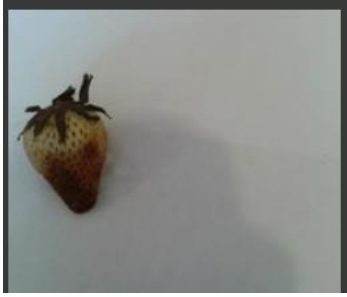
manipulation and random selection, respectively. And It has predicted correctly as cucumber_rotten.

```
# 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 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.

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., 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., 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., 1., 0., 0., 0.,
        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., 1., 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-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**

COMMENTS

FIRST AND FOREMOST, I SINCERELY GRATITUDE TO OUR ESTEEMED INISTITUTE SRI VASAVI DEGREE COLLEGE, FOR GIVING ME THIS OPPORTUNITY TO FULFILL OUR WARM DREAM TO BECOME A GRADUATE. OUR SINCERE GRADITUDE TO OUR SHORT TERM INTERNSHIP GUIDE SRI L LAKSHMI NARAYANA, LECTURER DEPARTMENT OF COMPUTER SCIENCE FOR TIMELY COOPERATION AND VALUABLE SUGGESTIONS WHILE CARRYING OUT THIS INTERNSHIP.

I EXPRESS MY SINCERE THANKS AND HEARTFUL GRATITUDE TO SRI L LAKSHMI NARAYANA, HOD IN COMPUTER SCIENCE FOR PERMITTING ME TO DO MY PROJECT INTERNSHIP.

I EXPRESS MY SINCERE THANKS AND HEARTFUL GRATITUDE TO SRI M RAMA KRISHNA, PRINCIPAL FOR PROVIDING A FAVOURABLE ENVIRONMENT AND SUPPORTING ME DURING THE DEVELOPMENT OF THIS INTERNSHIP.

THANK YOU,SMART BRIDGE

---- THINNULURU SRUTHI
TEAM LEADER

THE END

SIGNATURE OF THE HOD

SIGNATURE OF THE PRINCIPAL