INSURANCE FRAUD DETECTION USING MACHINE LEARNING

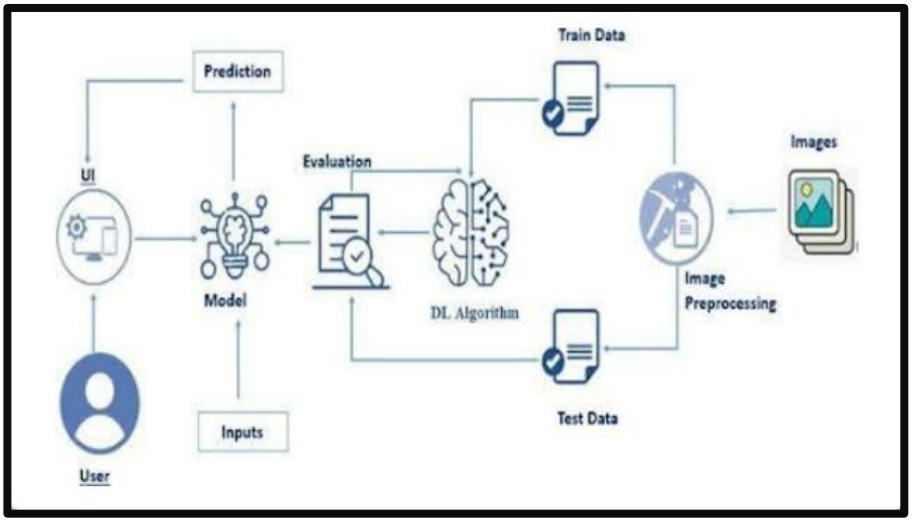
Project Hand-out, Faculty Development Program – NaanMudhalvan

SmartInternz
www.smartinternz.com

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.

Technical Architecture:



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

Prior Knowledge:

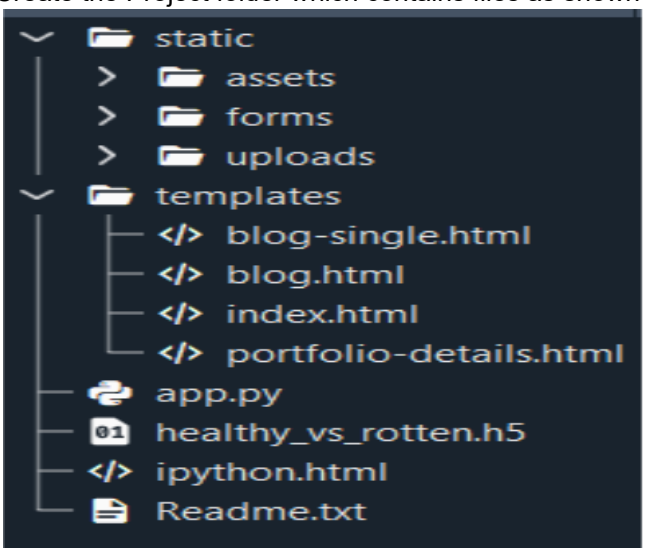
You must have prior knowledge of the following topics to complete this project.

- 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 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.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Problem Definition

Activity 2: Business requirements

A drug classification project can have a variety of business requirements, depending on the specific goals and objectives of the project. Some potential requirements may include:

·**Accurate Detection:** The system should correctly identify:

- Rotten fruits or vegetables
- Fresh fruits or vegetables
- The type of fruit or vegetable (e.g., apple, tomato, potato, etc.)

- **Flexibility:** The system should be able to work with different types of fruits and vegetables without complicated setup or major changes.
- **Compliance:** The sorting system should follow food safety standards and hygiene rules to ensure safe handling of produce.
- **Easy to Use:** The system should be simple to operate, with clear instructions and easy-to-understand controls for the workers.

Activity 4: Social or Business Impact.

Social Impact:- Improved food quality and safety by accurately detecting rotten fruits and vegetables, the sorting project ensures that only fresh produce reaches consumers. This leads to improved food quality, better health outcomes, and reduced chances of contaminated or spoiled food being sold in markets.

Business Model/Impact :- Reduced waste and increased efficiency by automatically identifying rotten items and sorting produce by type, the project helps reduce manual labor, minimize waste, and improve overall efficiency. This can lead to cost savings, higher productivity, and better profit margins for businesses involved in food processing or retail.

Milestone 2: Data Collection & 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.

Activity 1: 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 -p ~/.kaggle
!cp kaggle.json ~/.kaggle
!kaggle datasets download -d muhammad0subhan/fruit-and-vegetable-disease-healthy-vs-rotten
```

```
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 [02:23<00:00, 10.6MB/s]
100% 4.77G/4.77G [02:25<00:00, 35.3MB/s]
```

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

```
  inflating: Fruit And Vegetable Diseases Dataset/Potato__Rotten/rottenPotato (112).jpg
  inflating: Fruit And Vegetable Diseases Dataset/Potato__Rotten/rottenPotato (113).jpg
  inflating: Fruit And Vegetable Diseases Dataset/Potato__Rotten/rottenPotato (114).jpg
  inflating: Fruit And Vegetable Diseases Dataset/Potato__Rotten/rottenPotato (115).jpg
  inflating: Fruit And Vegetable Diseases Dataset/Potato__Rotten/rottenPotato (116).jpg
  inflating: Fruit And Vegetable Diseases Dataset/Potato__Rotten/rottenPotato (117).jpg
  inflating: Fruit And Vegetable Diseases Dataset/Potato__Rotten/rottenPotato (118).jpg
```

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

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(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.")
```

```
Strawberry__Healthy 200
Carrot__Healthy 200
Orange__Rotten 200
Mango__Rotten 200
```

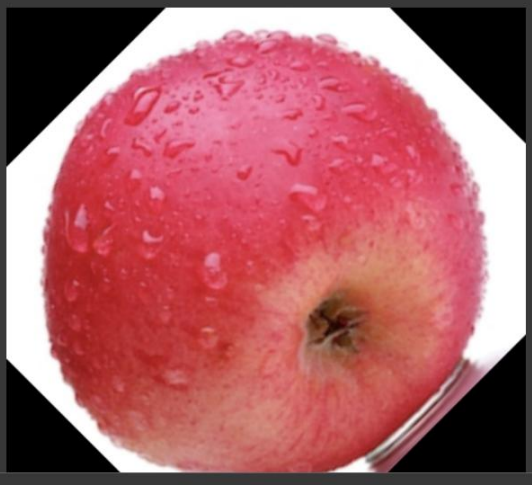
```
[ ] dataset_dir = '/content/output_dataset'
    train_dir = os.path.join(dataset_dir, 'train')
    val_dir = os.path.join(dataset_dir, 'val')
    test_dir = os.path.join(dataset_dir, 'test')

    IMG_SIZE = (224, 224)

    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=25,
        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)

    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=IMG_SIZE,
        batch_size=32,
        class_mode='categorical'
    )

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

    test_generator = val_test_datagen.flow_from_directory(
        test_dir,
        target_size=IMG_SIZE,
        batch_size=32,
        class_mode='categorical',
        shuffle=False
    )

    print(train_generator.class_indices)
    print(val_generator.class_indices)
    print(test_generator.class_indices)
```

Milestone 2: Exploratory Data Analysis

Activity 2.1: Data Visualization

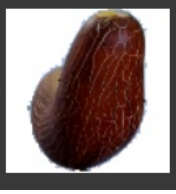
The provided Python code imports necessary libraries and modules for image manipulation. It selects a random image file from a specified folder path. Then, it displays the randomly selected image using IPython's Image module. This code is useful for showcasing random images from a directory for various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

```
[ ] import random
    from IPython.display import Image, display

    folder_path = '/content/output_dataset/test/Apple__Healthy'
    image_files = [f for f in os.listdir(folder_path) if f.endswith(('.jpg', '.png', '.jpeg'))]
    selected_image = random.choice(image_files)
    image_path = os.path.join(folder_path, selected_image)
    display(Image(filename=image_path))
```



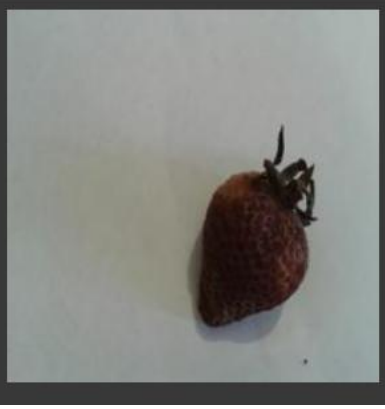
In the above code, I used class Apple_healthy 0 for prediction, This code randomly selects an image file from a specified folder (folder_path) containing JPEG, PNG, or JPEG files, and then displays the selected image using IPython's display function. It utilizes Python's OS and random modules for file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

```
folder_path = '/content/output_dataset/test/Cucumber__Rotten'
image_files = [f for f in os.listdir(folder_path) if f.endswith(('.jpg', '.png', '.jpeg'))]
selected_image = random.choice(image_files)
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.

```
folder_path = '/content/output_dataset/test/Strawberry__Rotten'
image_files = [f for f in os.listdir(folder_path) if f.endswith(('.jpg', '.png', '.jpeg'))]
selected_image = random.choice(image_files)
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.

Activity 2.2: 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

Milestone 3: Split Data

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,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
test_datagen = ImageDataGenerator(rescale=1./255)

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

```
Found 3359 images belonging to 28 classes.
Found 1119 images belonging to 28 classes.
```

Milestone 4: 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,weights="imagenet", input_shape=(224, 224, 3))

for layer in vgg.layers:
    print(layer)

print(len(vgg.layers))

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

x = Flatten()(vgg.output)
output = Dense(28, activation='softmax')(x)

vgg16 = Model(inputs=vgg.input, outputs=output)

vgg16.summary()
```

```
<Conv2D name=block3_conv2, built=True>
<Conv2D name=block3_conv3, built=True>
<MaxPooling2D name=block3_pool, built=True>
<Conv2D name=block4_conv1, built=True>
<Conv2D name=block4_conv2, built=True>
<Conv2D name=block4_conv3, built=True>
<MaxPooling2D name=block4_pool, built=True>
<Conv2D name=block5_conv1, built=True>
<Conv2D name=block5_conv2, built=True>
<Conv2D name=block5_conv3, built=True>
<MaxPooling2D name=block5_pool, built=True>
19
Model: "functional_3"
```

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808

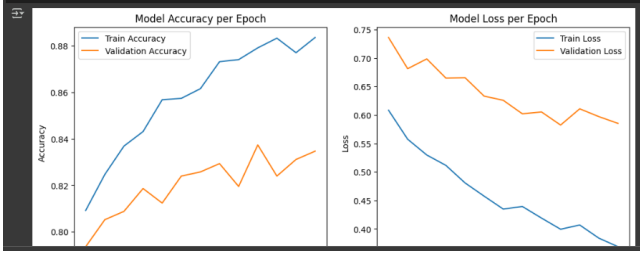
```python
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam

opt = Adam(learning_rate=0.0001)

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

# Compile the model
vgg16.compile(optimizer= opt, loss="categorical_crossentropy", metrics=["accuracy"])

# Train the model
history = vgg16.fit(
    train,
    validation_data=test,
    epochs=15,
    callbacks=[early_stopping]
)
```

```
Epoch 1/15
145/168 ━━━━━━━━━━━━━━━━━━━━ 10s 478ms/step - accuracy: 0.7913 - loss: 0.6479/usr/local/lib/python3.11/dist-packages/PIL/Image.py:1043: UserWarning: Palette images with Transparency expressed in bytes should be converted to RGBA images
  warnings.warn(
168/168 ━━━━━━━━━━━━━━━━━━━━ 98s 575ms/step - accuracy: 0.7938 - loss: 0.6428 - val_accuracy: 0.7936 - val_loss: 0.7363
Epoch 2/15
168/168 ━━━━━━━━━━━━━━━━━━━━ 97s 577ms/step - accuracy: 0.8225 - loss: 0.5455 - val_accuracy: 0.8052 - val_loss: 0.6814
Epoch 3/15
168/168 ━━━━━━━━━━━━━━━━━━━━ 101s 601ms/step - accuracy: 0.8329 - loss: 0.5181 - val_accuracy: 0.8088 - val_loss: 0.6986
Epoch 4/15
168/168 ━━━━━━━━━━━━━━━━━━━━ 95s 563ms/step - accuracy: 0.8436 - loss: 0.5124 - val_accuracy: 0.8186 - val_loss: 0.6650
Epoch 5/15
168/168 ━━━━━━━━━━━━━━━━━━━━ 92s 550ms/step - accuracy: 0.8603 - loss: 0.4480 - val_accuracy: 0.8123 - val_loss: 0.6657
Epoch 6/15
168/168 ━━━━━━━━━━━━━━━━━━━━ 96s 569ms/step - accuracy: 0.8596 - loss: 0.4478 - val_accuracy: 0.8239 - val_loss: 0.6333
Epoch 7/15
168/168 ━━━━━━━━━━━━━━━━━━━━ 94s 561ms/step - accuracy: 0.8737 - loss: 0.4030 - val_accuracy: 0.8257 - val_loss: 0.6259
Epoch 8/15
168/168 ━━━━━━━━━━━━━━━━━━━━ 144s 574ms/step - accuracy: 0.8814 - loss: 0.4110 - val_accuracy: 0.8293 - val_loss: 0.6022
Epoch 9/15
168/168 ━━━━━━━━━━━━━━━━━━━━ 94s 560ms/step - accuracy: 0.8707 - loss: 0.4160 - val_accuracy: 0.8195 - val_loss: 0.6054
Epoch 10/15
168/168 ━━━━━━━━━━━━━━━━━━━━ 95s 566ms/step - accuracy: 0.8909 - loss: 0.3828 - val_accuracy: 0.8374 - val_loss: 0.5824
Epoch 11/15
168/168 ━━━━━━━━━━━━━━━━━━━━ 101s 600ms/step - accuracy: 0.8767 - loss: 0.4196 - val_accuracy: 0.8239 - val_loss: 0.6109
Epoch 12/15
168/168 ━━━━━━━━━━━━━━━━━━━━ 95s 562ms/step - accuracy: 0.8845 - loss: 0.3712 - val_accuracy: 0.8311 - val_loss: 0.5972
Epoch 13/15
168/168 ━━━━━━━━━━━━━━━━━━━━ 94s 563ms/step - accuracy: 0.8862 - loss: 0.3733 - val_accuracy: 0.8347 - val_loss: 0.5854
```

Milestone 5: Performance Testing & Hyperparameter Tuning

First we check how well our model has been training in each epoch by plotting model loss and accuracy per epoch.

```python
import matplotlib.pyplot as plt

# Plot training & validation accuracy values
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



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

```
[ ] final_train_loss = history.history['loss'][-1]
    final_train_acc = history.history['accuracy'][-1]

    final_val_loss = history.history['val_loss'][-1]
    final_val_acc = history.history['val_accuracy'][-1]
    print("\nFinal Training Metrics:")
    print(f"Final Accuracy = {final_train_acc:.4f}")
    print(f"Final Loss = {final_train_loss:.4f}")

    print("\nFinal Validation Metrics:")
    print(f"Final Accuracy = {final_val_acc:.4f}")
    print(f"Final Loss = {final_val_loss:.4f}")
    # Evaluate the model on the test set
    # The evaluate method returns the loss and metrics
    test_loss, test_accuracy = vgg16.evaluate(test)

    # Print the test accuracy
    print(f"Accuracy on the test set: {test_accuracy:.4f}")
```

```
Final Training Metrics:
Final Accuracy = 0.8836
Final Loss = 0.3687

Final Validation Metrics:
Final Accuracy = 0.8347
Final Loss = 0.5854
56/56 ━━━━━━━━━━━━━━━━━━━━ 15s 258ms/step - accuracy: 0.8306 - loss: 0.6047
Accuracy on the test set: 0.8374
```

We got **83.74%** accuracy on test set !

Saving the model

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

```
vgg16.save('healthy_vs_rotten.h5')
print("Model Saved Successfully........")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving
Model Saved Successfully........
```

Milestone 6: Model Deployment

Application Building

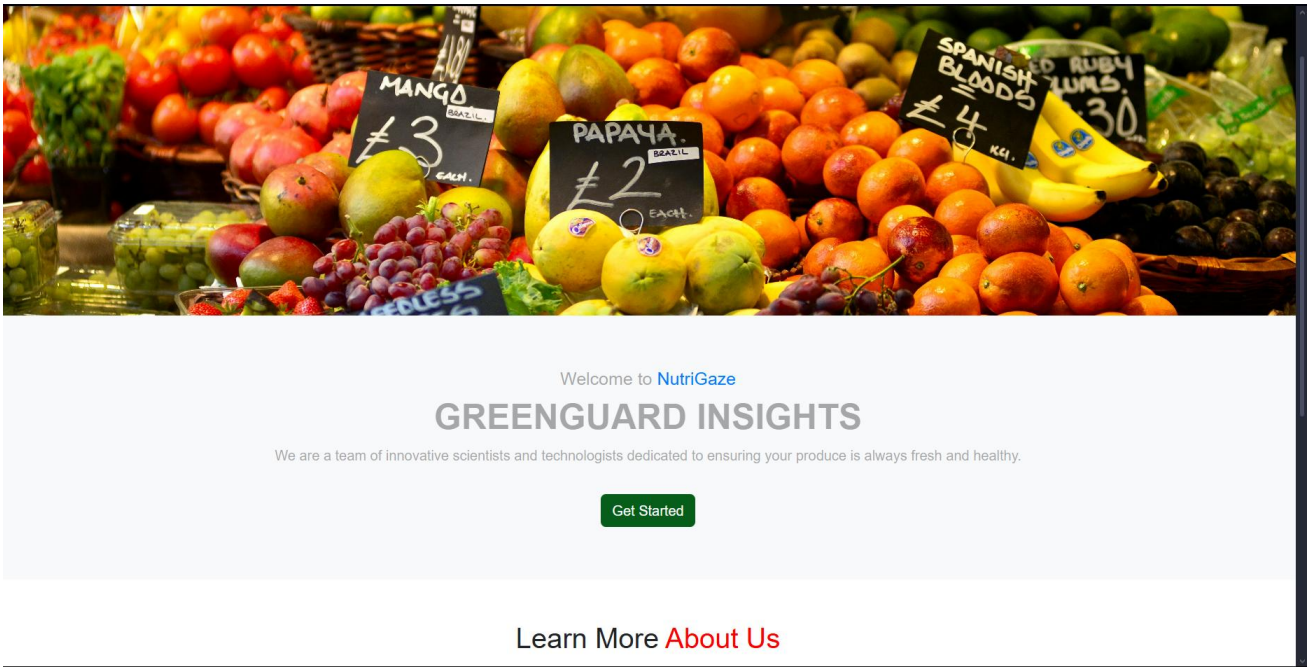
In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

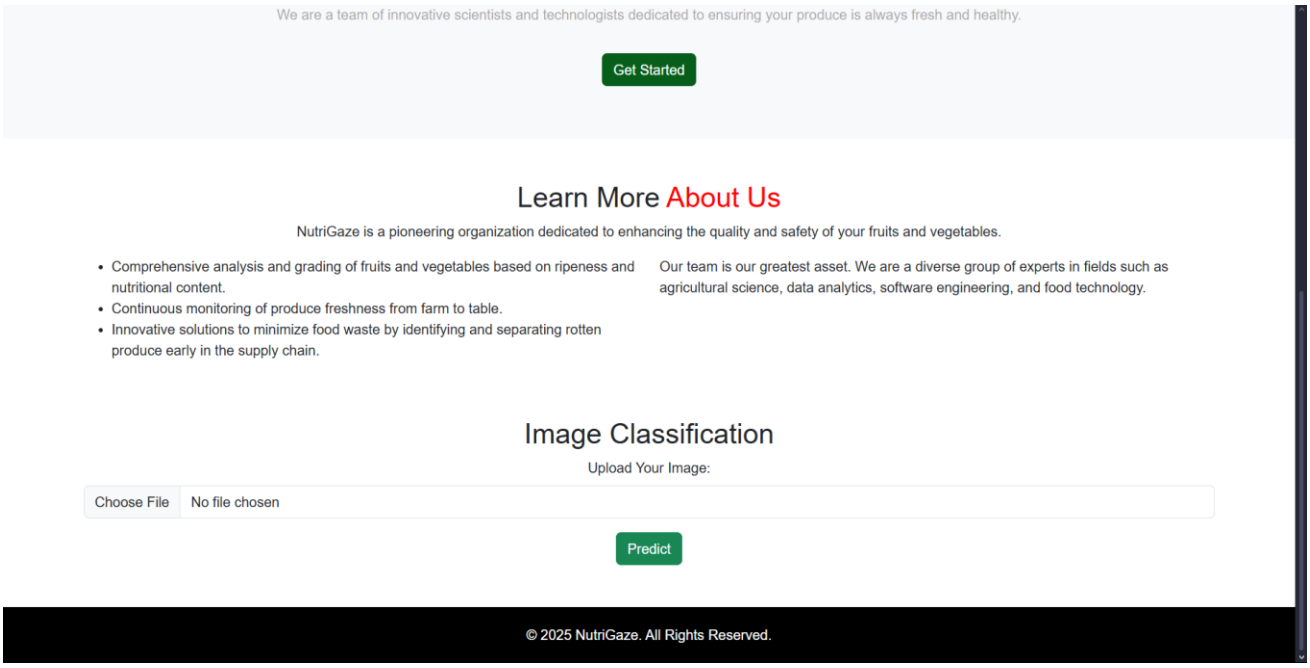
This section has the following tasks

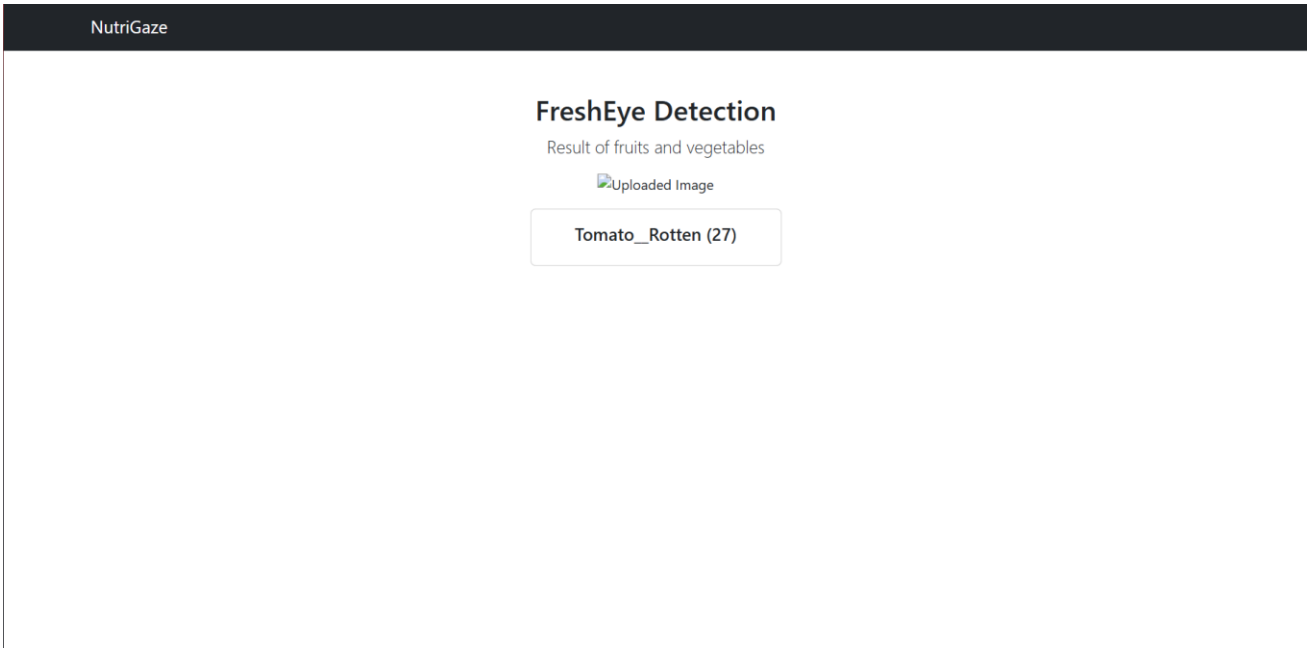
Building HTML Pages

Building server-side script

Activity 6.1: Building HTML Pages

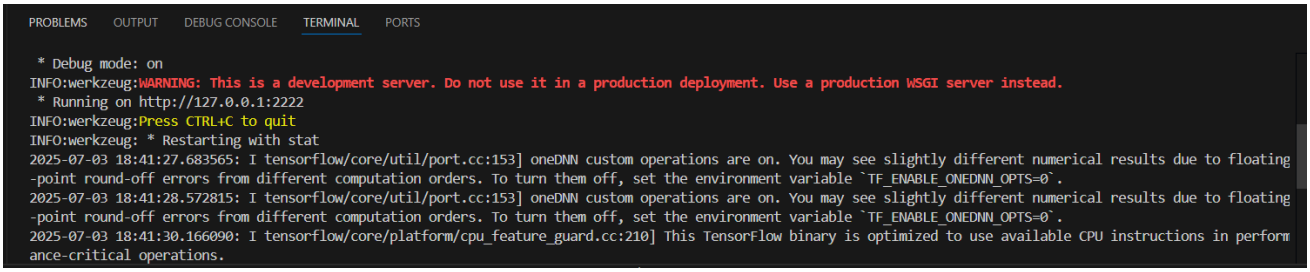






Python Code:

```python
from flask import Flask, render_template, request, jsonify, url_for, redirect
from tensorflow import keras
from keras._tf_keras.keras.preprocessing.image import load_img, img_to_array

from PIL import Image
import numpy as np
import os
import tensorflow as tf

# Initialize Flask app
app = Flask(__name__)

# Load the trained model
model = tf.keras.models.load_model('healthy_vs_rotten.h5')

# Route for home/index page
@app.route('/')
def index():
    return render_template("index.html")

# Route to handle prediction
@app.route('/predict', methods=['GET', 'POST'])
def output():
    if request.method == 'POST':
        file = request.files['file']
        file_path = os.path.join("static/uploads", file.filename)
        file.save(file_path)

        # Preprocess the image
        img = load_img(file_path, target_size=(224, 224))
        img = img_to_array(img)
        img = np.expand_dims(img, axis=0)
        img = img / 255.0  # Normalize

        # Prediction
        preds = model.predict(img)
        pred_index = np.argmax(preds)

        # Class labels (must match training labels)
        labels = [
            '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)',
```



Milestone 7: Project Demonstration & Documentation

Activity 1:- Video Demo Link

Activity 2:- Project Documentation-Step by step project development procedure