National College of Ireland

Configuration Manual:Machine Learning-Based Classification of Fruit Quality: Fresh and Rotten Fruit

MSc Research Project
Data Analytics

Pradeep Narayanaswamy
Student ID: X21199094

School of Computing
National College of Ireland

Supervisor: Shubham Shubnil

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Pradeep Narayanaswamy
Student ID:	X21199094
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Shubham Shubnil
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual:Machine Learning-Based Classification of Fruit Quality: Fresh and Rotten Fruit
Word Count:	691
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Pradeep Narayanaswamy
Date:	31st January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☑
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☑
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☑

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual:Machine Learning-Based Classification of Fruit Quality: Fresh and Rotten Fruit

Pradeep Narayanaswamy
X21199094

1 Introduction

The configuration manual outlines detailed information on the hardware and software prerequisites for the "Machine Learning-Based Classification of Fruit Quality: Fresh and Rotten Fruit". it also includes step-by-step instructions for project code execution.

2 System Prerequisites

The upcoming parts provide details on the essential prerequisites of hardware and software with the local configuration of the system development as shown in figure 1.

2.1 Hardware Prerequisites

- Processor: AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz or Intel.
- Installed RAM: 8.00 GB
- System type: 64-bit operating system, x64-based processor
- Storage: 500 GB HDD

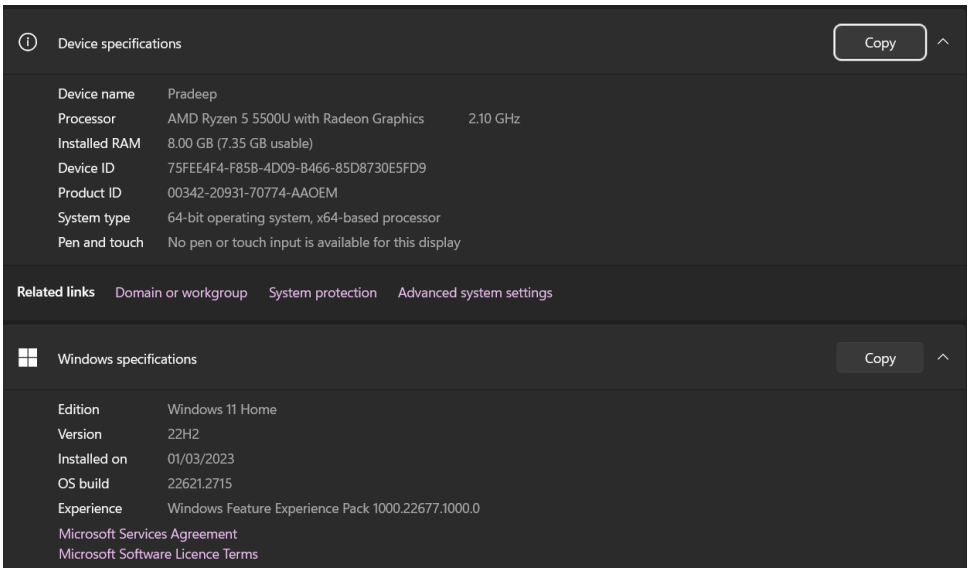


Figure 1: Devise Specification

2.2 Software Prerequisites

- Microsoft Excel: this is used for visualization and comparison of model performance
- Jupyter Notebook: Anaconda 3 offers different software tools for project execution, the jupyter notebook installed from Anaconda serves as the IDE for project development as shown in figure 2.

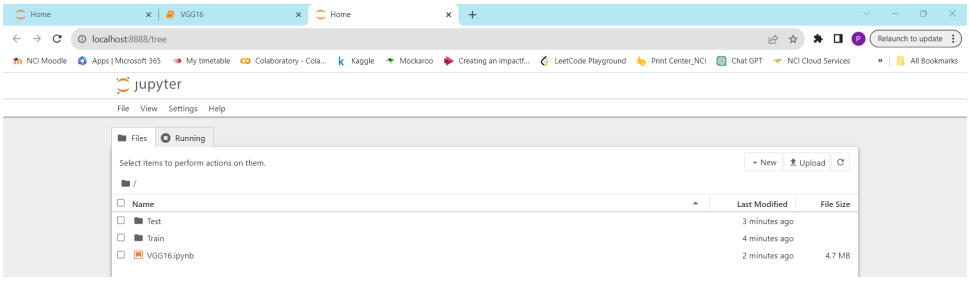


Figure 2: jupyter Notebook

- Python: The project uses Python 3.11.5 as shown in Figure **??** for project execution across various phases like Data analysis, Data Preprocessing, Model Training, and Evaluation.

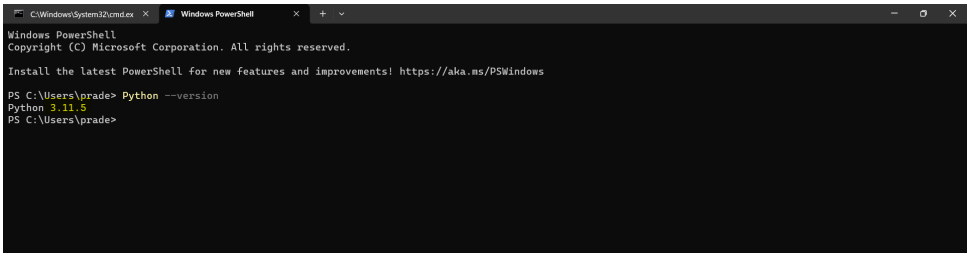


Figure 3: jupyter Notebook

3 Package Prerequisites

The essential Python packages were installed in Jupyter Notebook using the library pip, and the required packages are listed below:

- `os`: Provides operating system functionalities, facilitating file and directory operations.
- `random`: Enables randomization for various applications, such as shuffling data.
- `distutils`: Used for directory copy operations, aiding in the organization of project files.
- `cv2`: OpenCV library for image processing tasks, including manipulation and analysis.
- `numpy`: Essential for numerical operations, particularly efficient array handling.

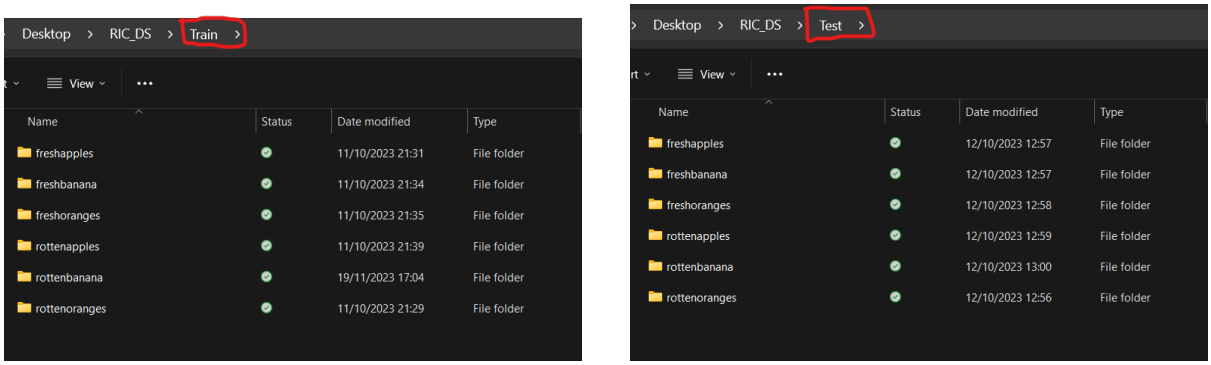
- `pandas`: Powerful data manipulation library, simplifying tasks like data cleaning and transformation.
- `matplotlib`: Widely used for data visualization and plotting in Python.
- `PIL`: Python Imaging Library, supporting various image handling tasks.
- `sklearn`: sci-kit-learn library, providing tools for machine learning tasks like classification and regression.
- `tensorflow`: Deep learning framework for building and training neural networks.

following commands are used to install packages:

```
pip install os
pip install random
pip install distutils
pip install opencv-python
pip install numpy
pip install pandas
pip install matplotlib
pip install pillow
pip install scikit-learn
pip install tensorflow
```

4 About Dataset

The dataset is obtained from" Fruit 360" from the Kaggle repository.[1]. The dataset contains two folders Test and Train as shown in Figure4 each folder contains six different sub-folders.



(a) Train Data Folder (b) Test Data Folder

Figure 4: Data Folder

[1]https://www.kaggle.com/datasets/moltean/fruits/data

5 Downloading and Execution of code

In the Artefacts section, only 100MB of data is permitted due to this limitation, the dataset and code artefacts are uploaded to OneDrive, and the link is shared with appropriate permissions.

Onedrive link for code and data: ClickForCodeData.

If the given previous link does not work due to an encoding issue, the short URL can be used: ClickForCodeData. By clicking on this link will open the folders as shown in Figure 5

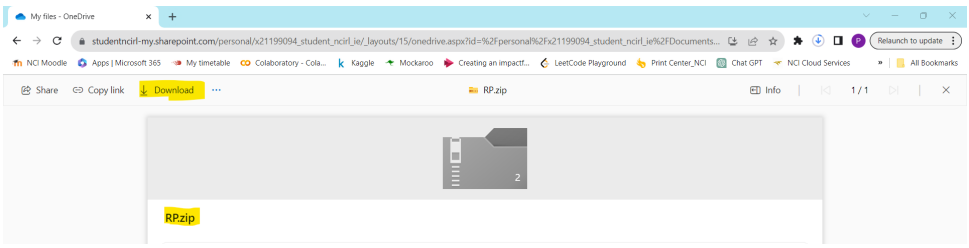


Figure 5: Zipped Folder Containing Code and Data Files

Clicking on the "Download" button, as shown in Figure 5 will initiate the download of both the code and data. The downloaded file is in a zipped format that must be unzipped.

To execute the code, initiate the Jupyter Notebook either through Anaconda or the command prompt(cmd). Navigate to the folder, open the command prompt(cmd), and execute the command "Python -m notebook." This action will launch a new tab in the default browser, as shown in Figure 6.

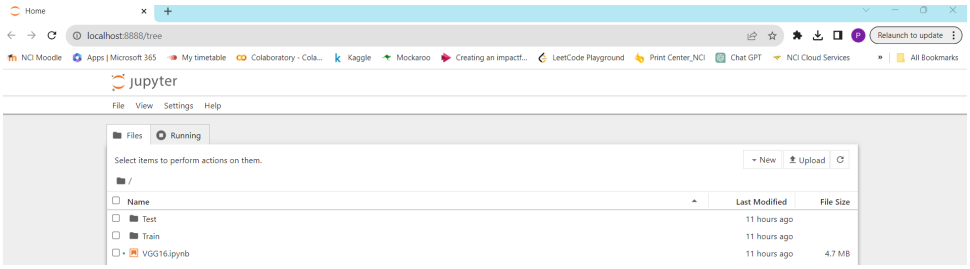


Figure 6: Running the code

The next step involves opening the file and executing the code by selecting the option Kernel -¿ Restart and Run All, as shown in Figure 7. This will execute all the code, including sections from importing libraries to evaluating models. During the code execution, some of the files are generated to store intermediate results

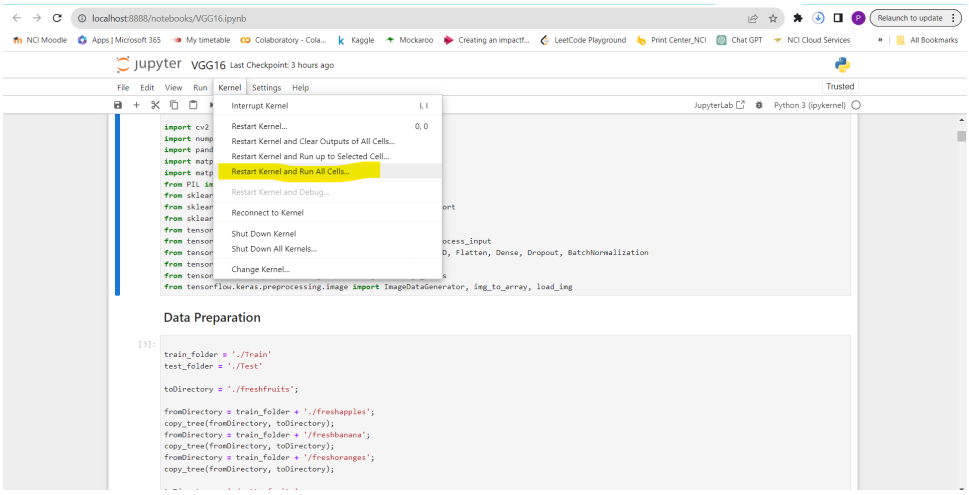


Figure 7: Running the code

The code is modularized for different functionalities, such as Data pre-processing, Data Augmentation, Model Training, and Evaluation. Each section is highlighted with specific details. Four models are trained for fruit classification, The code includes evaluations for each model.

Following are a few snippets of code from the project.

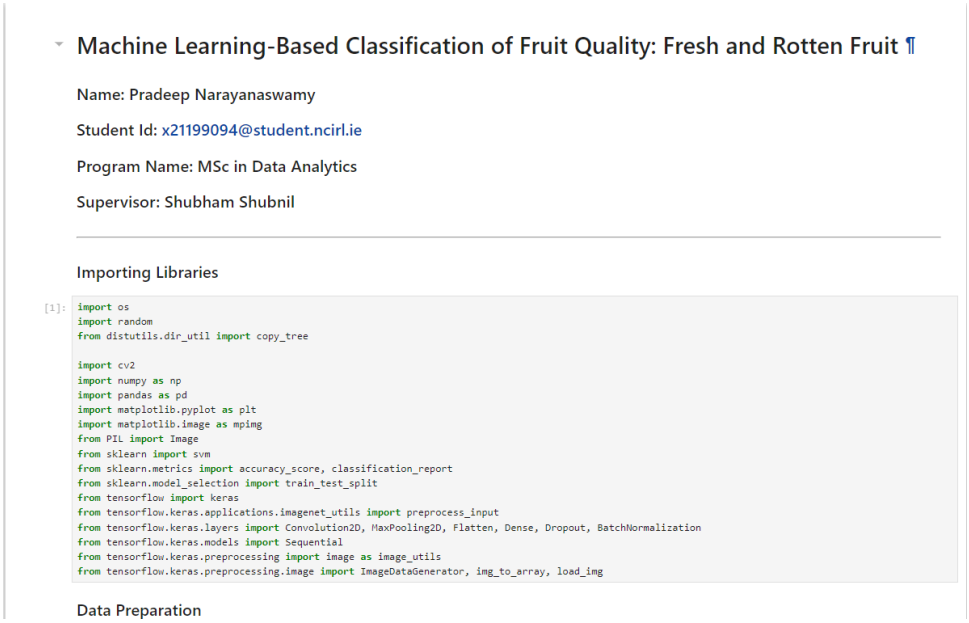


Figure 8: Importing Libraries

Importing Libraries

[1]:
```
import os
import random
from distutils.dir_util import copy_tree

import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from PIL import Image
from sklearn import svm
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from tensorflow import keras
from tensorflow.keras.applications.imagenet_utils import preprocess_input
from tensorflow.keras.layers import Convolution2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing import image as image_utils
from tensorflow.keras.preprocessing.image import ImageDataGenerator, img_to_array, load_img
```

Data Preparation

[3]:
```
train_folder = './Train'
test_folder = './Test'

toDirectory = './freshfruits';

fromDirectory = train_folder + './freshapples';
copy_tree(fromDirectory, toDirectory);
fromDirectory = train_folder + '/freshbanana';
copy_tree(fromDirectory, toDirectory);
fromDirectory = train_folder + '/freshoranges';
copy_tree(fromDirectory, toDirectory);

toDirectory = './rottenfruits';

fromDirectory = train_folder + '/rottenapples';
```

Figure 9: Data Preparation

Exploratory Data Analysis

[2]:
```
base_train_dir = './Train'
base_val_dir = './Test'
```

[3]:
```
train_dir = base_train_dir
test_dir = base_val_dir

fresh_apples_train_dir = os.path.join(train_dir, 'freshapples')
fresh_banana_train_dir = os.path.join(train_dir, 'freshbanana')
fresh_oranges_train_dir = os.path.join(train_dir, 'freshoranges')

#rotten
rotten_apples_train_dir = os.path.join(train_dir, 'rottenapples')
rotten_banana_train_dir = os.path.join(train_dir, 'rottenbanana')
rotten_oranges_train_dir = os.path.join(train_dir, 'rottenoranges')



fresh_apples_test_dir = os.path.join(test_dir, 'freshapples')
fresh_banana_test_dir = os.path.join(test_dir, 'freshbanana')
fresh_oranges_test_dir = os.path.join(test_dir, 'freshoranges')

#rotten
rotten_apples_test_dir = os.path.join(test_dir, 'rottenapples')
rotten_banana_test_dir = os.path.join(test_dir, 'rottenbanana')
rotten_oranges_test_dir = os.path.join(test_dir, 'rottenoranges')

fresh_apples_train_len=len(os.listdir(fresh_apples_train_dir))
fresh_banana_train_len=len(os.listdir(fresh_banana_train_dir))
fresh_oranges_train_len=len(os.listdir(fresh_oranges_train_dir))
rotten_apples_train_len=len(os.listdir(rotten_apples_train_dir))
rotten_banana_train_len=len(os.listdir(rotten_banana_train_dir))
rotten_oranges_train_len=len(os.listdir(rotten_oranges_train_dir))

print("Images of Training Dataset")
print("No. of train fresh apple images : ", fresh_apples_train_len)
print("No. of train fresh banana images : ", fresh_banana_train_len)
print("No. of train fresh orange images : ", fresh_oranges_train_len)
print("No. of train rotten apple images : ", rotten_apples_train_len)
```

Figure 10: Exploratory Data Analysis

Training Model

Model 1: Support Vector Machine (SVM)

Loading Dataset

```
[22]: def load_subset(folder, num_samples_per_class):
          images = []
          labels = []
          for class_folder in os.listdir(folder):
              class_path = os.path.join(folder, class_folder)
              sample_images = os.listdir(class_path)[:num_samples_per_class]
              for filename in sample_images:
                  img_path = os.path.join(class_path, filename)
                  img = cv2.imread(img_path)
                  img = cv2.resize(img, (100, 100))
                  images.append(img.flatten())
                  labels.append(class_folder)
          return np.array(images), np.array(labels)
```

```
[23]: train_folder = './trainimages'
      train_images, train_labels = load_subset(train_folder, num_samples_per_class=300)
```

Train Test Splitting

```
[24]: X_train, X_test, y_train, y_test = train_test_split(train_images, train_labels, test_size=0.2, random_state=42)
```

Model Preparation

```
[25]: svm_model = svm.SVC(kernel='linear')
```

Model Training

```
[26]: svm_model.fit(X_train, y_train)
```

```
[26]: ▾ SVC
      SVC(kernel='linear')
```

Figure 11: Model 1: Support Vector Machine (SVM)

Model 2: VGG 16

Model Preparation

```
[3]: base_model = keras.applications.VGG16(
         weights='imagenet',
         input_shape=(224, 224, 3),
         include_top=False);
```

```
[4]: base_model.trainable = False
```

```
[5]: inputs =  keras.Input(shape=(224, 224, 3))
     x = base_model(inputs, training=False)
     x = keras.layers.GlobalAveragePooling2D()(x)
     outputs = keras.layers.Dense(1,activation = 'sigmoid')(x)
     model = keras.Model(inputs, outputs)
```

Compiling Model

```
[6]: model.compile(loss=keras.losses.BinaryCrossentropy(from_logits=False), metrics=[keras.metrics.BinaryAccuracy()])
```

Performing Image Augmentation

```
[7]: datagen_train = ImageDataGenerator(
         samplewise_center=True,
         rotation_range=10,
         zoom_range=0.1,
         width_shift_range=0.1,
         height_shift_range=0.1,
         horizontal_flip=True,
         vertical_flip=True,
     )
```

```
[8]: img = load_img('./a_f001.png', target_size=(224, 224))
     img_array = img_to_array(img)
     img_array = np.expand_dims(img_array, axis=0)

     augmented_images = datagen_train.flow(img_array)
```

Figure 12: Model 2: VGG16

Model 3: VGG with Edge Detection

```
[4]:
import cv2
import numpy as np
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator

def apply_edge_detection(img):
    img = (img * 255).astype(np.uint8)
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    edges = cv2.Canny(gray, 50, 150)
    edges_rgb = cv2.cvtColor(edges, cv2.COLOR_GRAY2RGB)
    return edges_rgb
```

Sample Data Display After Edge Detection

```
[5]:
img_path = 'a_f001.png'
def display_edge_detected(img_path):
    original_img = image_utils.load_img(img_path, target_size=(224, 224))
    original_img_array = image_utils.img_to_array(original_img)
    edge_detected_img = apply_edge_detection(original_img_array)
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(original_img)
    plt.title('Original')
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(edge_detected_img)
    plt.title('Edge Detected')
    plt.axis('off')
    plt.show()
```

```
[6]:
display_edge_detected('a_f001.png')
```

Figure 13: Model 3 : VGG16 with Edge detection

Model 4: Multi-class Classification with VGG and Edge Detection

```
[19]:
def apply_edge_detection(img):
    img = (img * 255).astype(np.uint8)
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    edges = cv2.Canny(gray, 50, 150)
    edges_rgb = cv2.cvtColor(edges, cv2.COLOR_GRAY2RGB)
    return edges_rgb
```

```
[20]:
def custom_generator(generator):
    for batch_x, batch_y in generator:
        edge_batch_x = np.array([apply_edge_detection(img) for img in batch_x])
        yield edge_batch_x, batch_y
```

```
[21]:
vgg_model = keras.applications.VGG16(
    weights='imagenet',
    input_shape=(224, 224, 3),
    include_top=False
)
```

```
[22]:
vgg_model.trainable = False

input =  keras.Input(shape=(224, 224, 3))
x1 = vgg_model(input, training=False)
x1 = keras.layers.GlobalAveragePooling2D()(x1)
output = keras.layers.Dense(6, activation='softmax')(x1)
model = keras.Model(input, output)
```

```
[23]:
vgg_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 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
```

Figure 14: Model 4: Multi-class classification using VGG16+Edge detection