CLASSIFICATION OF TYPES AND CONDITIONS OF AN APPLE (FRUIT) USING DEEP LEARNING MODEL FOR CONSUMER ASSESSMENT

BY

Tin Vi Leed

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JUNE 2024

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: CLASSIFICATION OF TYPES AND CONDITIONS OF AN APPLE (FRUIT) USING DEEP LEARNING MODEL FOR CONSUMER ASSESSMENT

Academic Session: Jun 2024

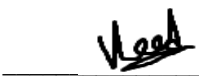
I TIN VI LEED

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,



(Author's signature)

(Supervisor's signature)

Address:
Lot 135-B5, Jalan Haji Ahmad,
25300, Kuantan, Pahang

Muhammad Syaiful Amri bin Suhaimi

Supervisor's name

Date: 26 August 2024

Date: 09/09/2024

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 26 August 2024

SUBMISSION OF FINAL YEAR PROJECT

It is hereby certified that ***Tin Vi Leed*** (ID No: ***21ACB04826***) has completed this final year project entitled "*Classification of types and conditions of an apple (fruit) using deep learning model for consumer assessment*" under the supervision of Muhammad Syaiful Amri Bin Suhaimi (Supervisor) from the Department of Computer Science, Faculty of Information and Communication Technology.

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

(Tin Vi Leed)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled "**CLASSIFICATION OF TYPES AND CONDITIONS OF AN APPLE (FRUIT) USING DEEP LEARNING MODEL FOR CONSUMER ASSESSMENT**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : 

Name : Tin Vi Leed

Date : 26 August 2024

ACKNOWLEDGEMENTS

At first, I would like to express my tremendous gratitude to my Final Year Project (FYP) supervisor, Dr Muhammad Syaiful Amri Bin Suhaimi for providing suggestions, guidance, and patience to answer all my questions regarding my project. Once again, thank you so much for spending your precious time and being patience to clear my doubt. You are a source of inspiration for me. It is an honour for me to get your assists when I encounter any difficulties. You are truly an amazing supervisor, and I am glad to be one of your students.

ABSTRACT

With the rapidly growing of consumer's demand for high quality fruits and the challenges they encountered in making informed decisions. Deep learning and mobile techniques come to the play to tackle these issues. Apples, one of the well-known fruits that contain of rich essential nutrients and are a popular choice for majorities. This project involved deep learning techniques and mobile technologies to implement a mobile application that will enable consumers to assess apple types and conditions quickly and accurately, thereby enhancing their shopping experiences and reducing time-consuming task of manual apple assessment. By addressing limitations of current practices, including overfitting and hardware requirements issues, this project will create a robust deep learning model incorporated with a user-friendly mobile application capable of generalizing in the real-world scenarios. In addition, the app would provide a convenient way to assess the apple simply by uploading an apple image. Ultimately, this project aims to reduce unnecessary food wastage, inspire future research and innovation in the field of fruit types and condition assessment.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
FYP	Final Year Project
VGG	Visual Geometry Group
UI	User Interface

Chapter 1
Introduction

1.1 Problem Statement and Motivation

In today's fast-paced world, consumers are increasingly seeking products that align with their preferences and meet high-quality standards. The variety of apple types and conditions available makes it challenging for consumers to make well-informed choices. Visual evaluation alone often falls short in providing accurate and comprehensive insights, therefore leading to this issue. Traditional methods such as relying on visual inspection, reading guides from internet, or drawing from personal experience, are time-consuming and may not taking into account for evolving consumer preferences and the diverse market. Furthermore, these conventional approaches are prone to errors due to subjective judgment and external factors that may influence decision-making. As a result, consumers may end up with apples that do not meet their expectations, which leads to dissatisfaction and unnecessary waste.

As the fruit industry evolves with new varieties and shifting consumer preferences, a novel approach is required to accurately determine apple types and conditions. Currently, there is no efficient and reliable method for differentiating between fresh and rotten apples. The absence of an easily accessible and objective way to assess apple freshness leaves consumers uncertain about the quality of their choices. This is where the integration of deep learning models and mobile technology can offer a transformative solution to this problem.

The motivation behind this project is the growing intersection of technology and evolving consumer preferences. A primary goal is to empower consumers to make informed purchasing decisions. In an era where personalization and customization are key, consumers desire products that closely match their preferences. By integrating a mobile app with deep learning capabilities, this project enables users to identify apples based on attributes like variety and freshness, giving them greater control over their choices.

This project is also motivated by the need to ensure consistent quality. In a competitive market, consumers tend to trust products that consistently meet their expectations. By utilizing deep

learning techniques, this project provides consumers with a reliable tool to navigate the wide array of apple varieties and conditions to enhance overall satisfaction.

Additionally, minimizing food waste is a critical aspect of this project's motivation. Visual inspection of apples often leads to inaccurate assessments which cause unavoidable wastage. By allowing consumers to make precise evaluations before purchasing, this project aims to reduce waste and promote responsible consumption and sustainability.

In summary, the motivation for this project is a multifaceted response to modern consumer preferences, technological advancements, and sustainability goals. By addressing the complexities of apple selection and evaluation, this project seeks to enhance the consumer experience, drive technological innovation, and support responsible consumption practices.

1.2 Objectives

- To develop a robust deep learning model to classify apple types and conditions in real world scenarios.
- To optimize for deep learning model deployment on mobile devices.
- To enhance user experience by implementing a user-friendly interface application.

1.3 Project Scope and Direction

The project scope includes data collection, model development, and mobile application implementation. The mobile app will feature a user-friendly interface for consumer assessment.

First of all, dataset collection is a crucial to enhance the model generalization. The dataset will be carefully collected to include four types of apple fruits (Cripps Red, Fuji, Granny Smith, Red Delicious), and various conditions (fresh, rotten, not apple). Following this, model development will be focusing on creating a deep learning model for apple fruit classification. Additionally, data augmentation techniques will also be explored during data preprocessing to improve model adaptability to real world scenarios. For example, lightning conditions, background changes, and different viewing angles. Another critical aspect is fine-tuning the model to ensure it is optimized for deployment on mobile devices, and achieving a balance

between accuracy and computational efficiency. Last but not least, the development of mobile application will provide a user-friendly interface that allows the consumers to take picture or upload image from gallery for apple types and conditions assessment to provide a convenient and satisfying shopping experience.

1.4 Contributions

This project has significant potential to contribute to society in various ways. Firstly, it addresses a pressing consumer need for accurate and efficient apple assessment methods. Secondly, it has the potential to reduce the number of apples that are discarded or left unused due to misjudgment. By addressing the challenge of accurately assessing apple conditions, this project not only benefits consumers but also offers advantages for retailers. The integration of such a solution into the retail environment can lead to higher customer satisfaction and a more efficient supply chain.

This project lies in its potential to transform how consumers interact with on of their most commonly consumed fruits, the apple. By offering a user-friendly mobile application powered by a deep learning model, consumers can make informed choices based on apple types and conditions. Furthermore, the project contributes to technological innovation by integrating deep learning models into mobile applications. This approach not only makes advanced technology more accessible for everyday consumers but also served as a proof of concept for the feasibility and effectiveness of using deep learning models for apple assessment thereby inspire future research and innovation in related fields.

1.5 Report Organization

This report is organized into 6 chapters: Chapter 1 Introduction, Chapter 2 Literature Review, Chapter 3 System Design, Chapter 4 System Design, Chapter 5 System Implementation, Chapter 6 System Evaluation and Discussion, and Chapter 7 Conclusion. The first chapter is the introduction of this project which includes problem statement and motivation, project objectives, project scope, project contribution, and report organization. The second chapter is the literature review on apple types and conditions classification approach. The third chapter is discussing the overall system design of this project. The fourth chapter is regarding the details on how the system workflow is being designed. Furthermore, the fifth chapter reports

the hardware and software setup, as well as the required setting and configuration. This chapter also demonstrates the system operations with screenshots and discuss about the implementation issues and challenges encountered. Following that, the sixth chapter covers the system evaluation and discussion, such as system testing, performance metrics utilized, testing setup, testing results, project challenges, and objectives evaluation. Lastly, the final chapter conclude the project works and provides recommendations to the project.

Chapter 2

Literature Review

This chapter focuses on past studies related to classification techniques and existing works specifically from year 2020 to 2023. The classification techniques applied in the project are reviewed in this chapter. Deep learning is a type of machine learning that involved neural network with multiple layers. These neural networks aim to mimic the functioning of the human brain by learning from large amounts of data, despite they are not powerful as the human brain. Unlike traditional machine learning, deep learning simplifies data preprocessing process. These algorithms can handle unstructured data such as images and text, and automating feature extraction, thereby reducing the reliance on human intervention [4].

2.1 Literature Review on Apple Fruit Types and Conditions Classification Approach

T. B. Kumar et al., [6] proposed a method to classify fruits conditions using convolutional neural networks (CNNs) to accurately differentiate between fresh and damaged fruits, such as oranges, apples, and bananas. The paper presents a detailed model architecture and methodology used to achieve this goal. The CNN model structure includes various layers, each serving a specific role in precise fruit classification. The initial phase consists of convolutional layers, which extract key features from input images. These layers work like filters, capturing important details such as edges, textures, and colors that indicate the quality of fruit. Figure 2.1.1 illustrates the role of these convolutional layers, which shows the basic CNN architecture used to detect these image features. Following this, the pooling layers are crucial for enhancing the model's efficiency. They reduce the size of the feature maps to prevent overfitting, which is a common issue in complex models. Figure 2.1.2 shows the proposed CNN structure and highlights how convolutional and pooling layers collaborate to improve the model’s performance. The block diagram shown in Figure 2.1.3 demonstrates a cyclic process that starts with 11-layer novel CNN model, which evaluates output images and accuracy, then processes data from a dataset downloaded from Kaggle.com. The subsequent steps involve image preprocessing by resizing the images, resulting in an input image of 256x256x3 for detection. A significant feature of this model is the utilization of dropout layers. These layers help address the overfitting problem by temporarily deactivating some neurons during training, making the model more adaptable and resilient to new situations beyond its training data. Implementing

this method requires the use of powerful frameworks such as TensorFlow and Keras. The model development relies on a well-prepared dataset containing various fruit classes for both fresh and rotten. This dataset is carefully prepared, including labelling and resizing, to ensure it is suitable for training and testing the model.

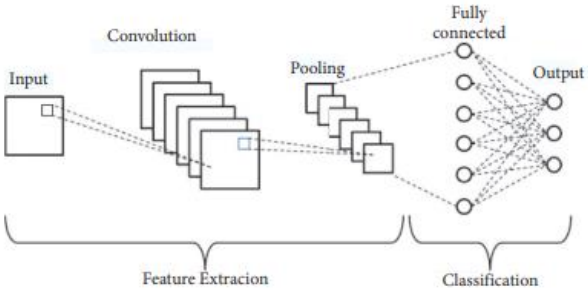


Figure 2.1.1 CNN Basic Architecture.

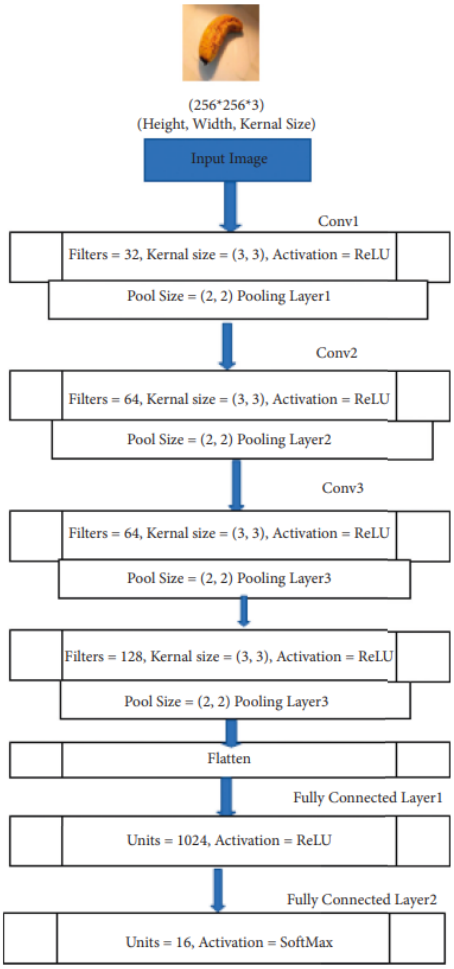


Figure 2.1.2 Proposed CNN model architecture.

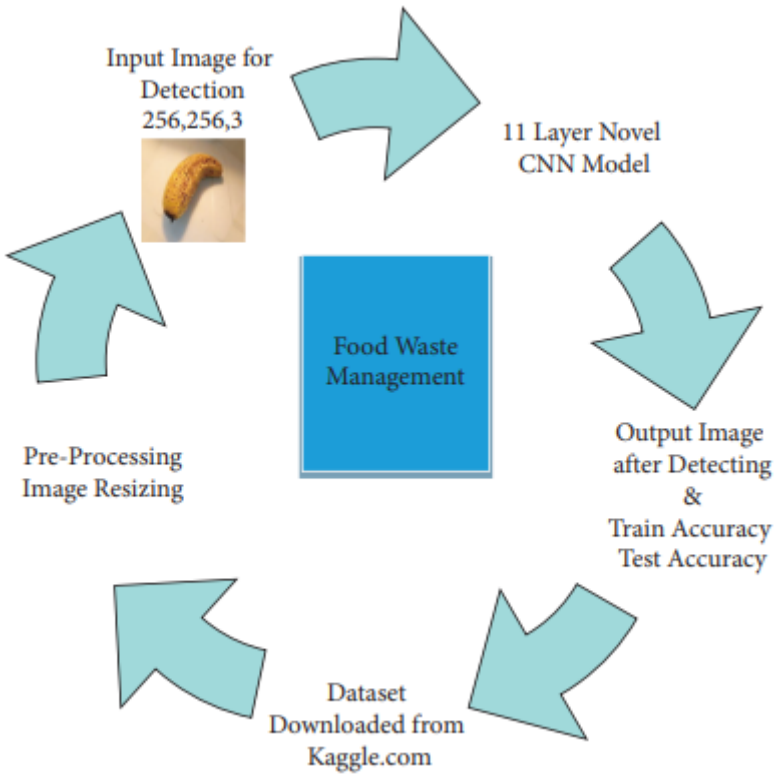


Figure 2.1.3 Block diagram of the proposed model.

Next, U.Shruthi et al. [7] proposed an innovative method for automatically classifying multiple varieties of apples using deep learning techniques. The primary goal of their research is to address the issues associated with labor-intensive and error-prone nature of manual apple classification processes. In order to achieve this, they introduced a specialized CNN architecture, specifically lightweight CNN model, tailored for apple types. The architecture is designed to extract relevant features from apple images, allowing for accurate differentiation among 14 different apple varieties as illustrated in Figure 2.1.4. The lightweight CNN model is composed of various components, including feature extraction and a classification parts. As shown in Figure 2.1.5, the feature extraction is enhanced through the innovative use of residual blocks, which capture the complex image features needed to distinguish different apple varieties. They hightlight that this approach outperforms existing architectures like VGG16, ResNet50, MobileNet, and EfficientNetB0 in terms of accuracy. The dataset utilized for training, validating, and testing their model consists of 7159 images representing 14 distinct categories of apple varieties. They recognize the inherent variability in image sizes and quality due to different sources and address this issue by preprocessing the images. This involves resizing all images to a standardized dimension of (160,160). Additionally, the preprocessing includes data augmentation techniques, such as flipping, rescaling, rotation, and zooming, which help expand and balance the training dataset. These augmentations increase the diversity

of input data, improving the model's ability to generalize and perform effectively on real-world images. Figure 2.1.6 provides example of apple image augmentation. The classification component of the proposed CNN architecture, as depicted in Figure 2.1.7, uses a fully connected layer that harnesses features extracted from residual blocks. This deep neural network comprises two hidden layers with 512 and 128 neurons respectively. The final output layer employs the softmax activation function to generate a probabilistic model of classifying each image.

Figure 2.1.4 Types of apples: 1. Granny Smith, 2. Pink lady, 3. Braeburn, 4.Red Delicious, 5. Cortland, 6. Rockit, 7. Rhode, 8. Honeycrisp, 9. Crimson, 10. Crispin, 11. Ambrosia, 12. Shizuka, 13. Paula Red, 14. Fuji.

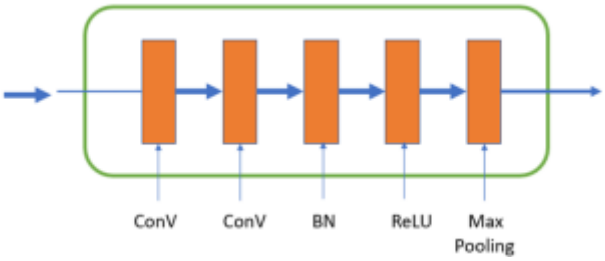


Figure 2.1.5 Residual block of proposed method.

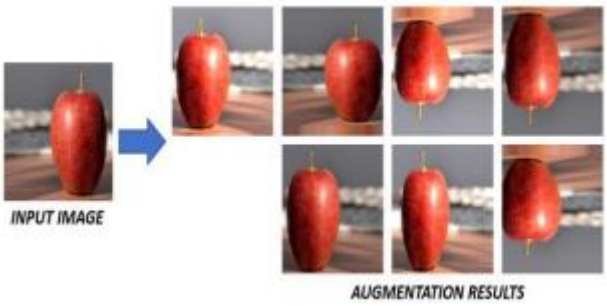


Figure 2.1.6 Example of apple image augmentation.

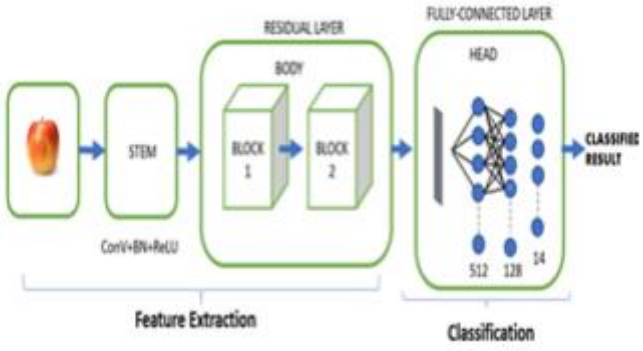


Figure 2.1.7 Proposed CNN architecture's classification component.

In the study of S.Chakraborty et al.[8], an automated method was proposed to identify rotten fruits, such as apples, oranges, and bananas, using deep CNN and MobileNetV2. Initially, the dataset contains both fresh and rotten fruits and was sourced from Kaggle. Data augmentation steps were then applied to enhance the dataset's diversity and suitability for the model. Each image in the dataset was resized to a standardized dimension of 256x256 pixels and normalized. The data augmentation procedure included horizontal flipping, zooming, rotating, and shearing as illustrated in Figure 2.1.8. Following the data preprocessing and augmentation, the proposed deep CNN utilized three convolution layers with the ReLU activation function for classification and image recognition. Additionally, Max Pooling and Average Pooling operations were employed to capture important features and patterns. The working process of the model is shown in Figure 2.1.9 and 2.1.10. For the second model, MobileNetV2, was integrated into the system to classify fruit quality. Notably, the layer of MobileNetV2 taken from TensorFlow were frozen and set to non-trainable. Trainable layers were then added on top of the base model, and these new layers were trained on the dataset. Lastly, a dropout layer was also applied to address potential overfitting issues.

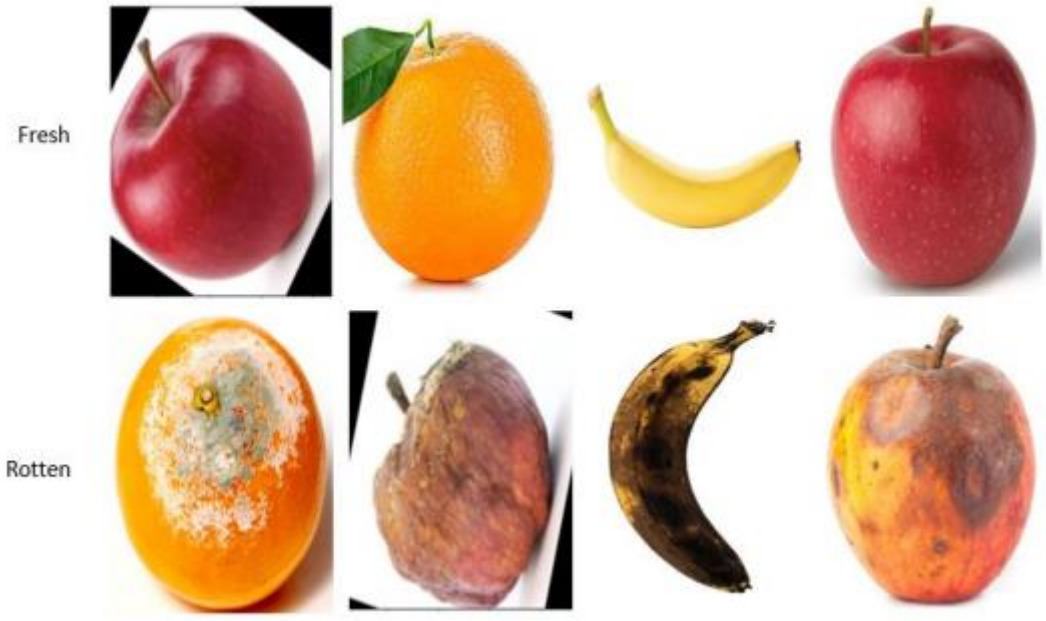


Figure 2.1.8 Example of augmented dataset images.

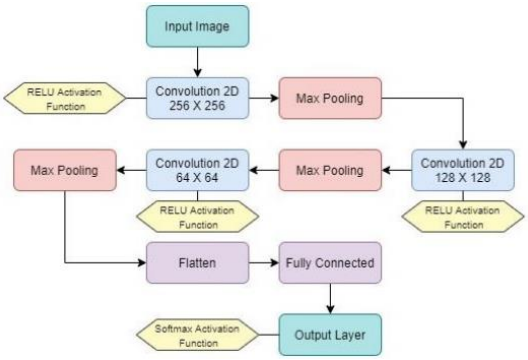


Figure 2.1.9 CNN model working process with Max pooling operation.

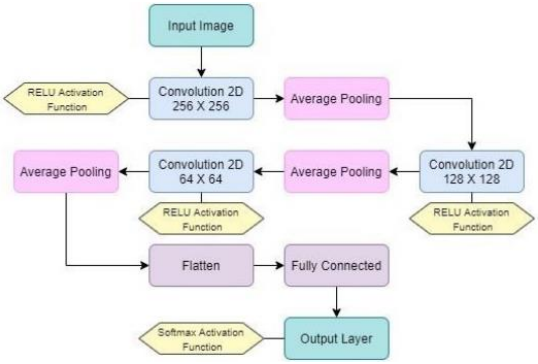


Figure 2.1.10 CNN model working process with AVERAGE pooling operation.

2.2 Critical Remarks on Previous Works

In paper [7], the authors introduced a CNN model designed to detect fresh and damaged fruits, specifically apples, oranges, and bananas. While the model achieves strong classification performance for each fruit, its performance may vary when applied to a broader range of fruit types. Additionally, the paper did not address key factors that could impact the model's performance, such as varying lighting conditions, viewing angles, and backgrounds. These limitations underscore the need for further research in their future work to improve the model's generalizability.

In paper [8], the authors presented a lightweight CNN model for classifying different types of apples. However, the model's reliance on a limited or relatively small dataset could compromise its performance and make it prone to overfitting. Although the proposed model outperforms other existing models, its performance is still questionable as the paper did not evaluate or test the model under different background and lighting conditions in real-world scenarios. Additionally, the lightweight CNN model is designed to be computationally efficient, which has limited memory and processing power compared to other CNN model, therefore there is a tradeoff between performance and accuracy.

In paper [6], the authors proposed a deep learning method to determine the freshness of fruits, including apples, oranges, and bananas, using a CNN to capture features from input fruit images with Max Pooling, Average Pooling, and MobileNetV2 for classification. However, the proposed CNN model is highly sensitive to changes in lighting conditions and color variations in the input fruit images, which may result in inaccurate detection of rotten fruits. Moreover, the complex textures and irregular shapes of fruits present challenges to the CNN architecture. The paper also raises concerns about overfitting due to the limited dataset used.

One of the common limitations across existing deep learning models is their dependency on data, because it is “data hungry” which require access to large datasets for effective training. The availability of such datasets is crucial for ensuring accurate apple fruit classification. However, the process of dataset collection and preparation can be time-consuming [5], [6], [7], [8]. Another limitation is the hardware requirements, especially when deploying deep learning models on mobile devices. While the lightweight CNN models proposed in papers [5] and [7] are designed to be computationally efficient, they still require a certain level of hardware resources. Running deep learning models on most low-end and mid-range mobile devices can be challenging due to limited computational power, memory constraints, and battery life [11].

In summary, the limitations identified in previous studies and existing practices include data limitations, data dependency, generalization capabilities for classifying apple types and conditions in real-world scenarios, concerns about model overfitting, and hardware requirements when deploying deep learning models on mobile devices.

Chapter 3

System Methodology

3.1 System Design Diagram

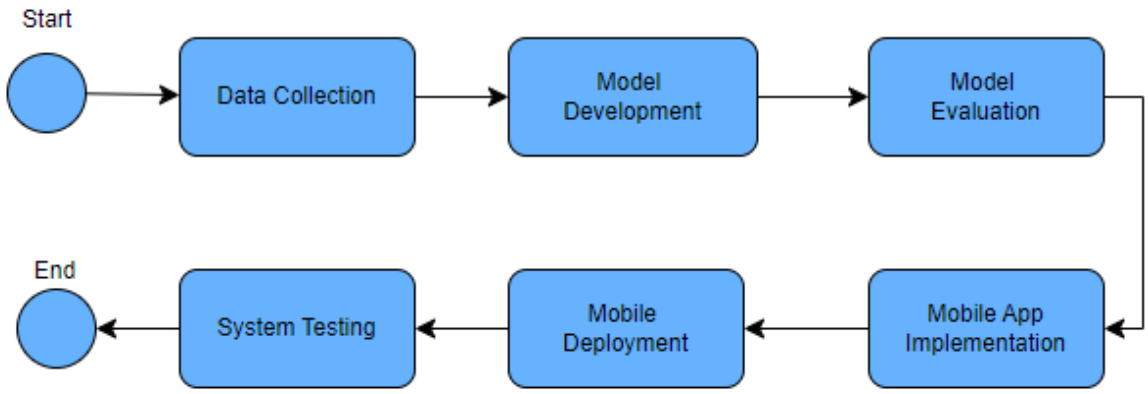


Figure 3.1.1 System general workflow.

Figure 3.1.1 shows the general workflow of the project (FYP2). It begins with data collection, involving the collection of a diverse dataset consisting of images of different apple types (Cripps Red, Fuji, Granny Smith, Red Delicious, not apple) and conditions (fresh, rotten). Images of apple types are collected and labeled manually, while images of apple conditions are sourced from Kaggle. Following this, data augmentation techniques are applied to enhance the dataset's diversity and robustness, incorporating methods such as rotating, shifting, zooming, flipping, shearing, resizing, and normalizing. In the next phase, three different models are defined (self-defined custom CNN, MobileNetV2 with pretrained weights, MobileNetV2 without pretrained weights) and trained on the prepared datasets. These models are evaluated using a classification report and confusion matrix to assess the model's capability in correctly classifying apple types and conditions. The model with best performance is then chosen for mobile deployment. Finally, the system undergoes system testing to ensure its usability to enhance consumer experience.

3.2 Tools and Technologies Involved

3.2.1 Software

Table 3.2.1 Software involved.

Name	Version	Description
Jupyter Notebook	7.0.2	A web-based interactive computing environment utilized to write and run Python code for deep learning model development.
Visual Studio Code	1.92.2	A code editor used to write and debug Dart code for mobile app development with Flutter.
Android Studio	2024.1	An IDE used to run the mobile phone emulator to test and deploy model to the mobile application.
Flutter	3.22.2	A UI toolkit used to build natively compiled mobile application

3.2.2 Coding/Programming Language

1. Python 3.11.4
2. Dart 3.4.3

Table 3.2.2 Coding/Programming language involved.

Name	Version	Description
Python	3.11.4	Used in Jupyter Notebook for deep learning model development.
Dart	3.4.3	Used in Visual Studio Code with Flutter to develop the mobile application.

3.2.3 Hardware

Table 3.2.3 Specifications of desktop.

Description	**Specifications**
Processor	AMD Ryzen 5 5600 6-Core Processor
Operating System	Microsoft Windows 11 Home
Graphic	NVIDIA GeForce GTX 1660 SUPER 6GB VRAM
Memory	16GB DDR4 RAM 3200 MHz

Storage	256 GB NVME M.2 SSD 1TB SATA SSD

Table 3.2.4 Specifications of smartphone

Description	Specifications
Model	Xiaomi 13T
Processor	Mediatek Dimensity 8200 Ultra
Graphic	Mali-G610 MC6
Operating System	MIUI Global 14.0.5 (Android 13)
Memory	8GB RAM
Storage	256 GB

3.3 Project Timeline

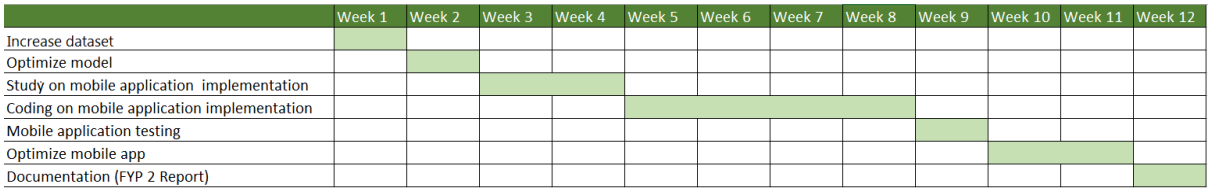


Figure 3.3.1 FYP2 timeline.

Chapter 4

System Design

4.1 System Block Diagram

This section includes the system block diagram for both deep learning model development and project overall workflow.

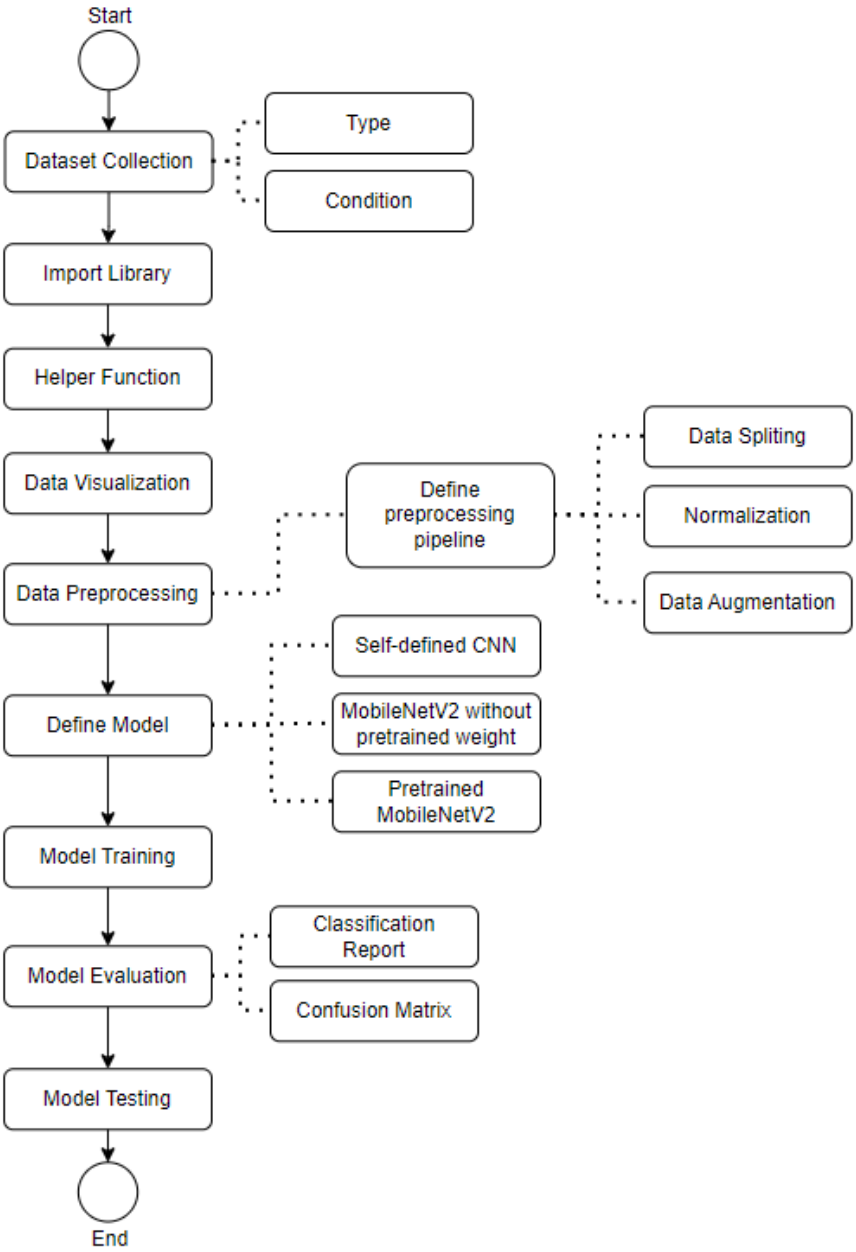


Figure 4.1.1 Deep learning model development workflow.

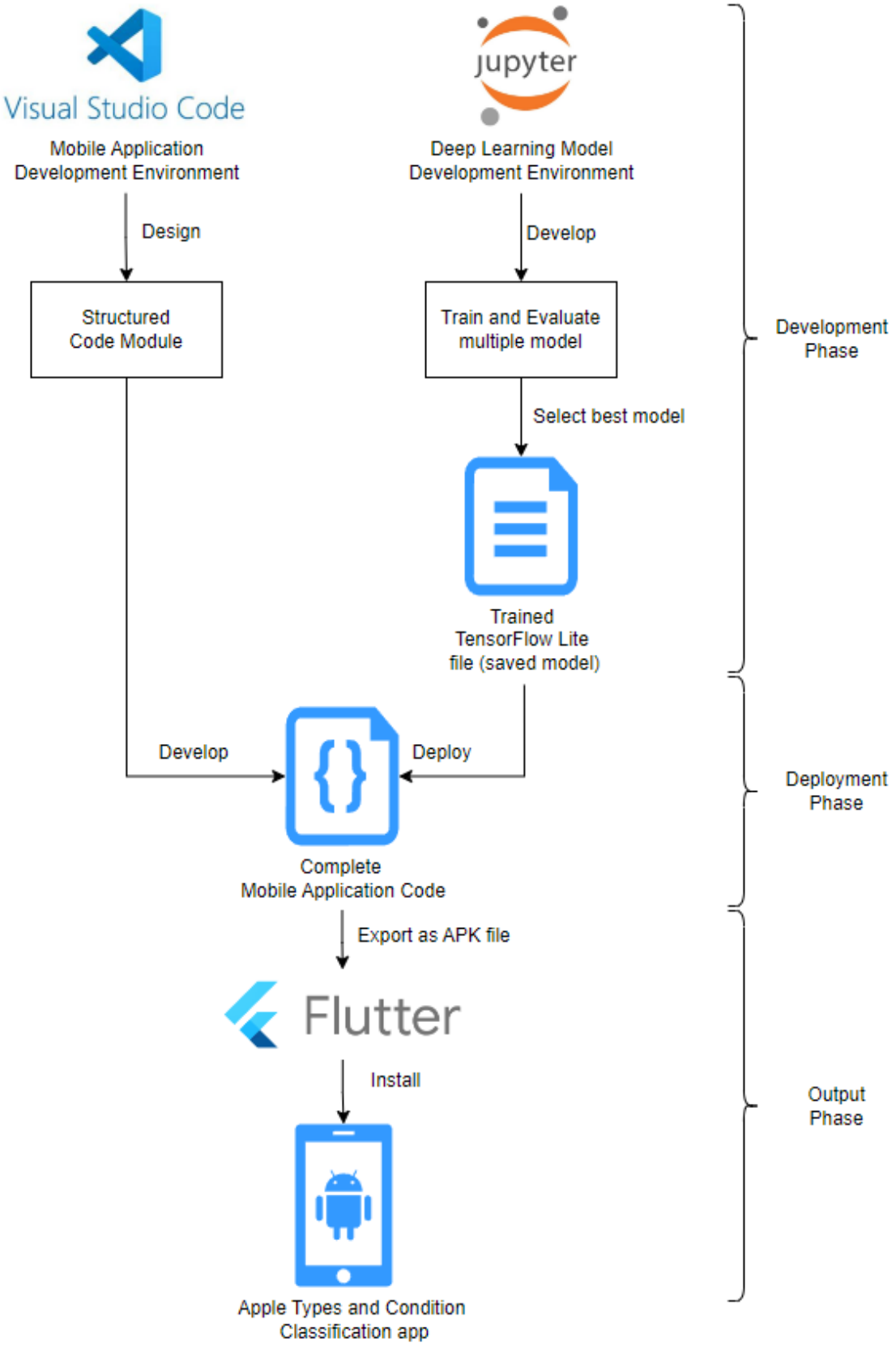


Figure 4.1.2 FYP2 overall workflow.

4.2 System Components Specifications (Deep Learning Model Development)

Different models were trained to classify apple types and conditions. Each model was trained separately in different Jupyter Notebook files to ensure optimal training configurations and to facilitate easier debugging and experimentation.

4.2.1 Dataset Collection

A total of 2,181 images were collected for classifying apple types, and 1,823 images were collected for classifying apple conditions. The images for apple types were manually collected and labelled into five classes, such as Cripps Red, Fuji, Granny Smith, Red Delicious, and Not Apple. In contrast, the images for apple conditions were downloaded from Kaggle, which included pre-labelled annotations. These images were categorized into two classes which include fresh and rotten. As shown in Figure 4.2.1 and Figure 4.2.2, all images were stored on desktop PC to facilitate further processing and model development. Additionally, Figure 4.2.3 shows the example of apple types images.

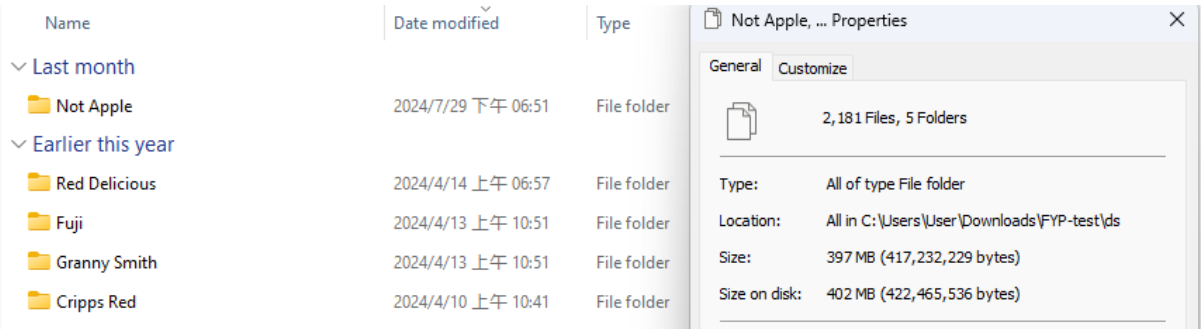


Figure 4.2.1 Apple types images.

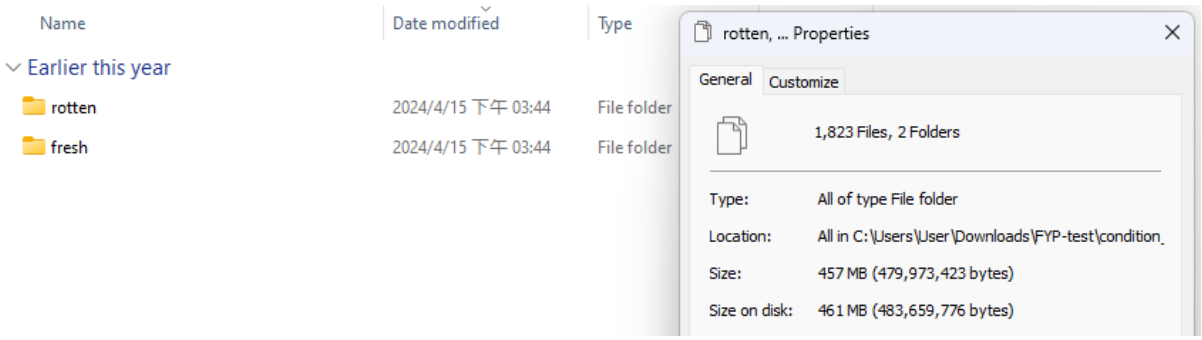


Figure 4.2.2 Apple conditions images.

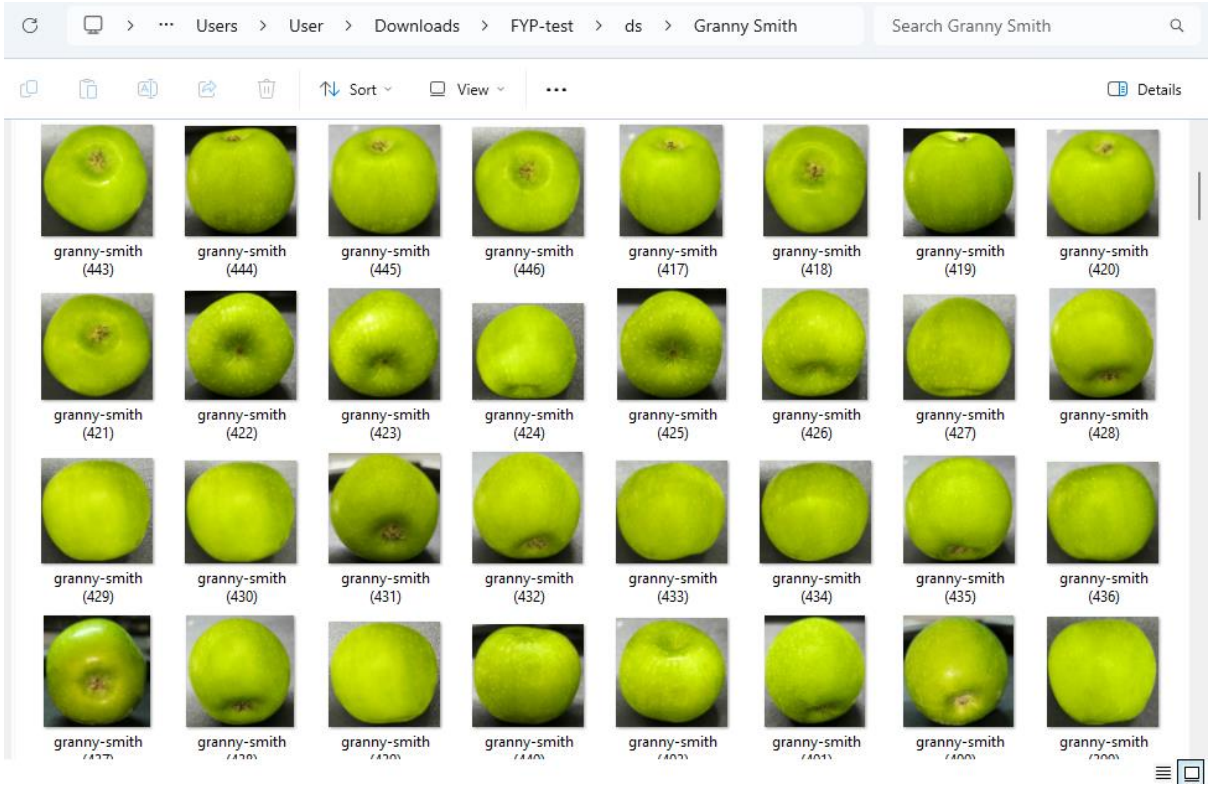


Figure 4.2.3 Example of apple types images (Granny Smith).

4.2.2 Import Library

As shown in Figure 4.2.4, necessary TensorFlow libraries are imported for following implementation.

```
# tensorflow libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Activation, BatchNormalization, Conv2D, Flatten, Dense, DepthwiseConv2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import Callback, EarlyStopping,ModelCheckpoint, ReduceLROnPlateau

# system libraries
import os
import random
from pathlib import Path

# visualization libraries
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import cv2
import seaborn as sns

import pandas as pd
import numpy as np

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, make_scorer
from sklearn.model_selection import train_test_split
```

Figure 4.2.4 Imported TensorFlow library.

4.2.3 Helper Function

There are 4 constant variables declared as shown in Figure 4.2.5.

- DATASET_PATH, this constant variable is assigned the value of "C:\\Users\\User\\Downloads\\FYP-test\\ds\\", indicating the location of the data set on the local computer storage.
- BATCH_SIZE, the batch size is configured to 32, meaning each training session will process 32 images in a single batch.
- EPOCHS, this refers to the number of training iterations the model will go through.
- TARGET_SIZE, this is defined as the tuple of (224, 224), specifying the target dimension for resizing images before they are fed into the model.

```
# Define constants
DATASET_PATH = "C:\\Users\\User\\Downloads\\FYP-test\\ds\\"
BATCH_SIZE = 32
EPOCHS = 20
TARGET_SIZE = (224, 224)
```

Figure 4.2.5 Constant variables.

Figure 4.2.6 shows the defined helper functions.

```
def walk_through_dir(dir_path):
  for dirpath, dirnames, filenames in os.walk(dir_path):
    print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
```

```
def load_dataset(file):
    data = []
    class_labels = []
    for label, class_folder in enumerate(os.listdir(file)):
        class_labels.append(class_folder)
        class_path = os.path.join(file, class_folder)
        if os.path.isdir(class_path):
            for image_file in os.listdir(class_path):
                image_path = os.path.join(class_path, image_file)
                # Append (file_path, label) tuple to data list
                data.append((image_path, label))
    return data, class_labels
```

Figure 4.2.6 Helper functions.

The “walk_through_dir” function displays the total number of subdirectories and images within each directory. The “load_dataset” function then lists all the images located in the “DATASET_PATH” along with their corresponding class labels. Figure 4.2.7 illustrates the outputs of both functions.

```
# Walk through each directory for apple types
walk_through_dir(DATASET_PATH);
```

```
There are 5 directories and 0 images in 'C:\Users\User\Downloads\FYP-test\ds\'.
There are 0 directories and 390 images in 'C:\Users\User\Downloads\FYP-test\ds\Cripps Red'.
There are 0 directories and 463 images in 'C:\Users\User\Downloads\FYP-test\ds\Fuji'.
There are 0 directories and 446 images in 'C:\Users\User\Downloads\FYP-test\ds\Granny Smith'.
There are 0 directories and 441 images in 'C:\Users\User\Downloads\FYP-test\ds\Not Apple'.
There are 0 directories and 441 images in 'C:\Users\User\Downloads\FYP-test\ds\Red Delicious'.
```

```
image, labels = load_dataset(DATASET_PATH)
print(image)
print(labels)
```

```
[('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (1).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP
-test\\ds\\Cripps Red\\cripps-red (10).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red
(100).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (101).jpg', 0), ('C:\\Users\\Use
r\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (102).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Crip
ps Red\\cripps-red (103).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (104).jpg',
0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (105).jpg', 0), ('C:\\Users\\User\\Downloads
\\FYP-test\\ds\\Cripps Red\\cripps-red (106).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\crip
ps-red (107).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (108).jpg', 0), ('C:\\Use
rs\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (109).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\d
s\\Cripps Red\\cripps-red (11).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (110).j
pg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (111).jpg', 0), ('C:\\Users\\User\\Down
loads\\FYP-test\\ds\\Cripps Red\\cripps-red (112).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red
\\cripps-red (113).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (114).jpg', 0),
('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (115).jpg', 0), ('C:\\Users\\User\\Downloads\\FY
P-test\\ds\\Cripps Red\\cripps-red (116).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-r
ed (117).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (118).jpg', 0), ('C:\\Users
\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (119).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds
\\Cripps Red\\cripps-red (12).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\ds\\Cripps Red\\cripps-red (120).jp
```

```
# Walk through each directory for apple conditions
walk_through_dir(DATASET_PATH)
```

```
There are 2 directories and 0 images in 'C:\Users\User\Downloads\FYP-test\condition_ds\'.
There are 0 directories and 911 images in 'C:\Users\User\Downloads\FYP-test\condition_ds\fresh'.
There are 0 directories and 912 images in 'C:\Users\User\Downloads\FYP-test\condition_ds\rotten'.
```

```
image, labels = load_dataset(DATASET_PATH)
print(image)
print(labels)
```

```
[('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (1).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP
-test\\condition_ds\\fresh\\fresh (1).png', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh
(10).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (10).png', 0), ('C:\\Users\\User
\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (100).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition
_ds\\fresh\\fresh (100).png', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (101).jpg',
0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (101).png', 0), ('C:\\Users\\User\\Downloads
\\FYP-test\\condition_ds\\fresh\\fresh (102).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh
\\fresh (102).png', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (103).jpg', 0), ('C:\\Us
ers\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (103).png', 0), ('C:\\Users\\User\\Downloads\\FYP-test
\\condition_ds\\fresh\\fresh (104).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (10
4).png', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (105).jpg', 0), ('C:\\Users\\User
\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (105).png', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition
_ds\\fresh\\fresh (106).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (106).png',
0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (107).jpg', 0), ('C:\\Users\\User\\Downloads
\\FYP-test\\condition_ds\\fresh\\fresh (107).png', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh
\\fresh (108).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (108).png', 0), ('C:\\Us
ers\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (109).jpg', 0), ('C:\\Users\\User\\Downloads\\FYP-test
\\condition_ds\\fresh\\fresh (109).png', 0), ('C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\fresh\\fresh (1
```

Figure 4.2.7 Output of helper functions.

4.2.4 Data Visualization

To gain a clearer understanding of each class and identify any class imbalances, bar plots are utilized to visualize the distribution of apple types and conditions in the dataset. Figure 4.2.8 shows the distribution for each class. Additionally, sixteen images are randomly selected and

displayed in a grid format, where accompanied by their respective class labels. Rather than displaying sixteen images separately for both apple types and conditions, Figure 4.2.9 presents a combination of both apple types and conditions within the same set of sixteen images.

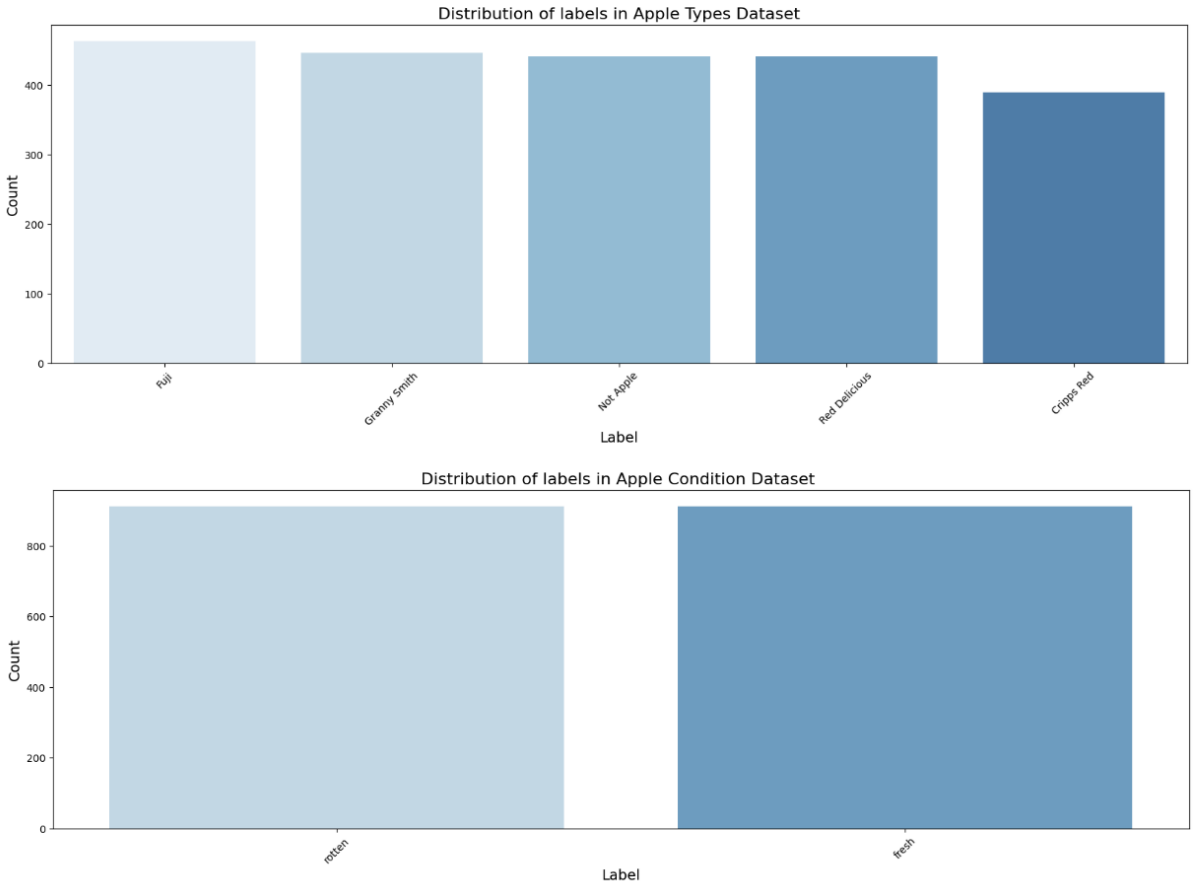


Figure 4.2.8 Distribution of each class label.

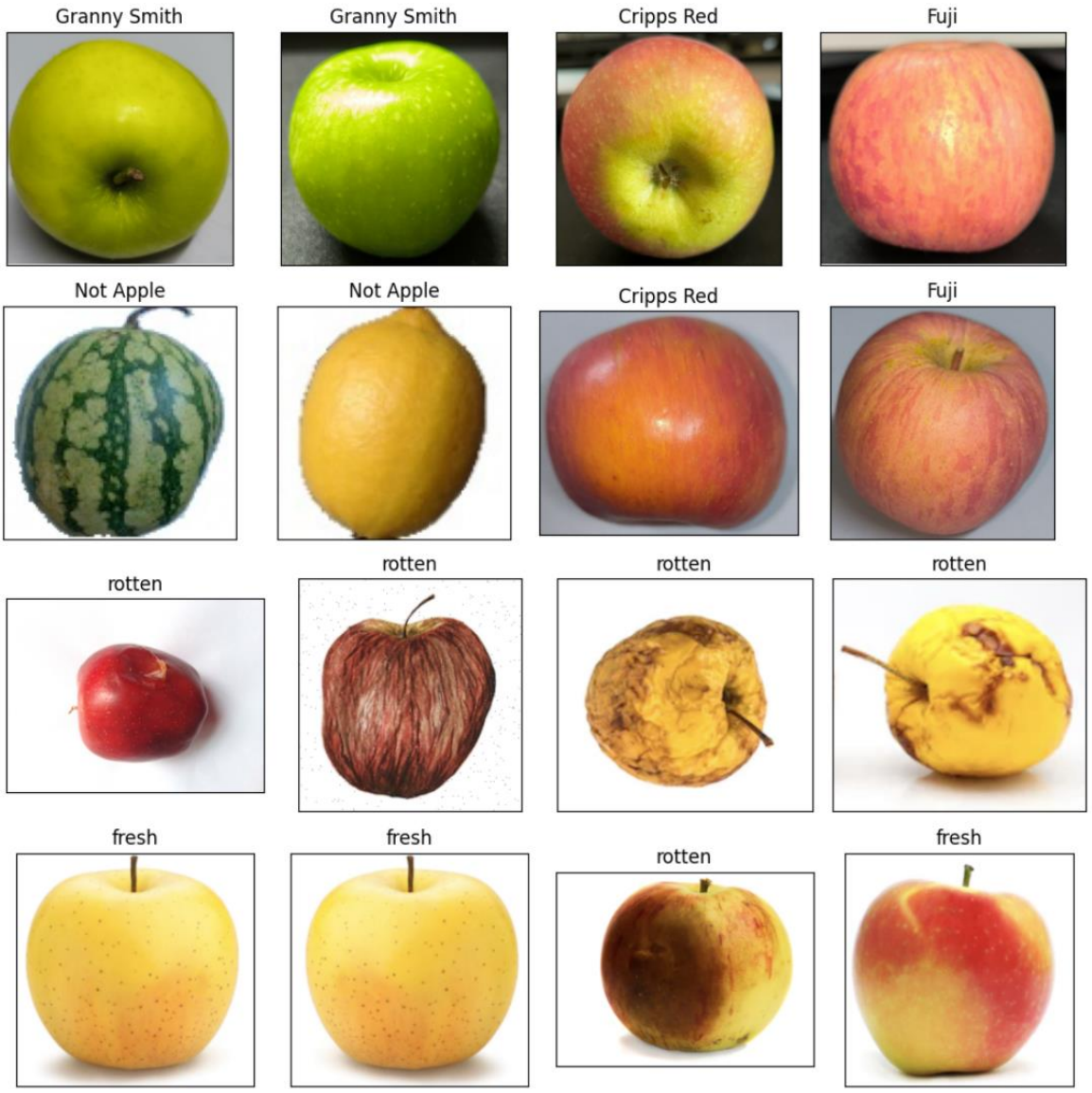


Figure 4.2.9 Random selected images of both apple types and conditions.

4.2.5 Data Preprocessing

There are total of 2181 for apple types and 1,823 images for apple conditions were collected for this project. To ensure the input data is suitable for training the deep learning model, a preprocessing pipeline was established, incorporating data splitting, normalization, and augmentation. The dataset is divided into a training set (80%) and a testing set (20%). Additionally, 20% of the training set is further split into a validation set. Figure 4.2.10 displays the number of samples in each subset, while Figure 4.2.11 illustrates the data preprocessing methods using the "ImageDataGenerator". Notably, two separate "ImageDataGenerator"

instances are defined for apple type and condition classification, with the class mode set to categorical and binary, respectively.

```
Found 1396 validated image filenames belonging to 5 classes.
Found 348 validated image filenames belonging to 5 classes.
Found 437 validated image filenames belonging to 5 classes.
Class Indices: {'Cripps Red': 0, 'Fuji': 1, 'Granny Smith': 2, 'Not Apple': 3, 'Red Delicious': 4}
Number of training samples: 1396
Number of validation samples: 348
Number of testing samples: 437

Found 1167 validated image filenames belonging to 2 classes.
Found 291 validated image filenames belonging to 2 classes.
Found 365 validated image filenames belonging to 2 classes.
Class Indices: {'fresh': 0, 'rotten': 1}
Number of training samples: 1167
Number of validation samples: 291
Number of testing samples: 365
```

Figure 4.2.10 Number of samples in each subset.

```python
# Set up data generators for training, validation, and testing
train_datagen = ImageDataGenerator(
    rescale=1./255,             # Rescale pixel values to [0, 1]
    validation_split=0.2,       # Split 20% of the data for validation
    rotation_range=20,          # Randomly rotate images by up to 20 degrees
    width_shift_range=0.1,      # Randomly shift images horizontally by up to 10%
    height_shift_range=0.1,     # Randomly shift images vertically by up to 10%
    shear_range=0.1,            # Apply shear transformation with intensity up to 10%
    zoom_range=0.1,             # Randomly zoom into images by up to 10%
    horizontal_flip=True,       # Randomly flip images horizontally
    fill_mode='nearest'         # Strategy for filling in newly created pixels after rotation or shifting
)
```

Figure 4.2.11 Data preprocessing approaches.

4.2.6 Define Model

This project defines three models, a self-defined customizable CNN model, a pretrained MobileNetV2, and a MobileNetV2 without pretrained weight.

The self-defined CNN model begins with a convolutional layer containing 32 filters and a 3x3 kernel size. Each convolutional layer is followed by a max-pooling layer to downsample the feature maps. Additional convolutional layers are then added with increasing numbers of filters (64 and 128) to capture more complex features, with max-pooling layers after each. The feature maps are subsequently flattened and passed through dense layers with ReLU activation functions. Dropout layers are also included after each dense layer for regularization to prevent overfitting. Finally, an output layer with an appropriate activation function is added to classify

the input images into one of the predefined classes. The model is compiled using the Adam optimizer with a suitable loss function (sparse categorical crossentropy) and accuracy metrics. For classifying apple types, a softmax activation function is used in the output layer, while a sigmoid activation function is used for classifying apple conditions. Figure 4.2.12 presents the model summary of the proposed CNN.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 222, 222, 32)      896

 max_pooling2d (MaxPooling2  (None, 111, 111, 32)      0
 D)

 conv2d_1 (Conv2D)           (None, 109, 109, 64)      18496

 max_pooling2d_1 (MaxPoolin  (None, 54, 54, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 52, 52, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 26, 26, 128)       0
 g2D)

 flatten (Flatten)           (None, 86528)             0

 dense (Dense)               (None, 256)               22151424

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 128)               32896

 dropout_1 (Dropout)         (None, 128)               0

 dense_2 (Dense)             (None, 4)                 516

=================================================================
Total params: 22278084 (84.98 MB)
Trainable params: 22278084 (84.98 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Figure 4.2.12 Model summary of proposed self-defined CNN.

Two different settings of MobileNetV2 models are also defined in this project, one with the base model initialized with pretrained weights from ImageNet, and the other without pretrained weights. Both MobileNetV2 models start with a base architecture that consists of a series of depthwise separable convolutional layers. A global average pooling layer is then applied to reduce each feature map to a single vector. The models proceed to a dense output layer with an appropriate activation function, similar to the self-defined CNN model shown above in Figure 4.2.12. The softmax activation function is used for classifying apple conditions. The models are also compiled using the Adam optimizer, with suitable loss functions (sparse categorical crossentropy) and accuracy metrics. Figure 4.2.13 shows the model summary of proposed MobileNetV2.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Func  (None, 7, 7, 1280)        2257984
 tional)

 global_average_pooling2d (  (None, 1280)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 5)                 6405

=================================================================
Total params: 2264389 (8.64 MB)
Trainable params: 6405 (25.02 KB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

Figure 4.2.13 Model summary of proposed MobileNetV2.

4.2.7 Model Training

This project implements a standardized model training procedure across all models. It starts with the callbacks, which includes "EarlyStopping", "ModelCheckpoint", and "ReduceLROnPlateau". Each component is configured with specific parameters. "EarlyStopping" monitors the validation loss with a patience of five epochs, meaning that the training process will terminate early if there is no improvement in validation loss for five consecutive epochs. "ModelCheckpoint" is set up to save the best model based on validation loss. Additionally, "ReduceLROnPlateau" is configured with a patience of two and a minimum learning rate of 0.00001, which means it will reduce the learning rate if the validation loss does not improve for two epochs and will not lower the learning rate below its minimum setting. However, "ReduceLROnPlateau" is considered optional as the model may not significantly benefit from dynamic learning rate reduction.

Following the setup of callbacks, the model is trained using the ".fit()" function, utilizing both training and validation data generators. The model is trained for a fixed number of twenty epochs with a batch size of thirty-two. The number of steps per epoch is dynamically calculated based on the number of samples and batch size. The specific settings and configurations of these training procedures, as well as the setup of the callbacks, are illustrated in Figure 4.2.14, which provides a detailed overview of the model training process.

```
# Define callbacks
callbacks = [
    EarlyStopping(patience=5, monitor='val_loss', verbose=1),
    ModelCheckpoint('custom_CNN_type1.h5', save_best_only=True, monitor='val_loss', mode='min'),
    # ReduceLROnPlateau(factor=0.1, patience=2, min_lr=0.00001, verbose=1),
]

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    callbacks=callbacks
)
```

Figure 4.2.14 Model training settings.

4.2.8 Model Evaluation

Each model's performance is assessed using a standardized method. This involves running evaluations on the test set using the ".evaluate()" function. The training and validation loss were plotted over each epoch to visualize the model's learning process, as shown in Figures 4.2.15 to 4.2.20. This provided valuable insights into how well the model was learning over time and helped identify overfitting or underfitting issues. If the training loss decreased while the validation loss remained high or increased, it indicated overfitting, where the model performed well on training data but poorly on unseen data. Conversely, a high training loss suggested underfitting, where the model failed to learn the data's features effectively. A detailed discussion of the model evaluation results, including specific metrics (accuracy, train loss, validation loss), classification report, confusion matrix, and insights, will be provided in Chapter 6.

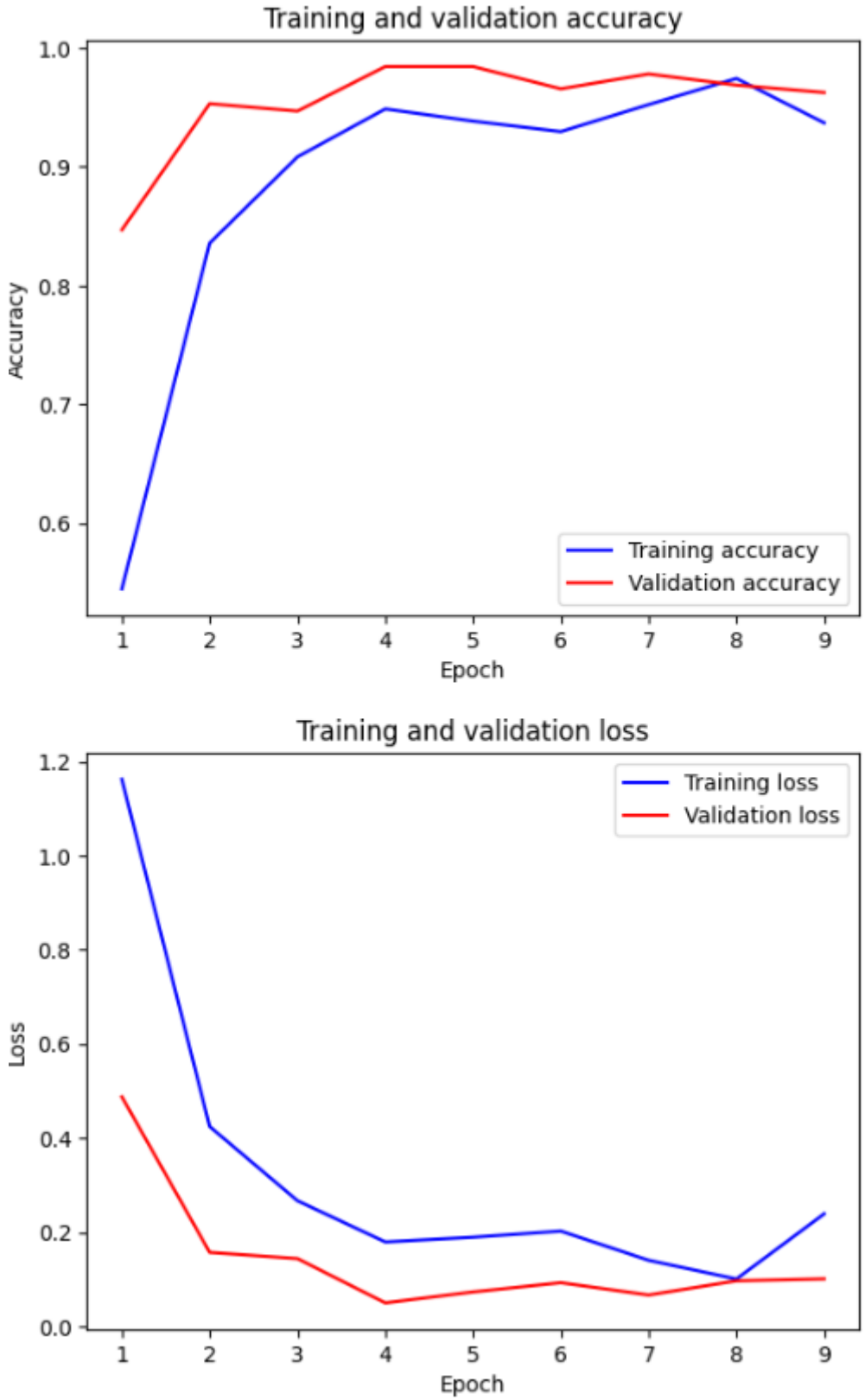


Figure 4.2.15 Self-defined custom CNN (apple types).

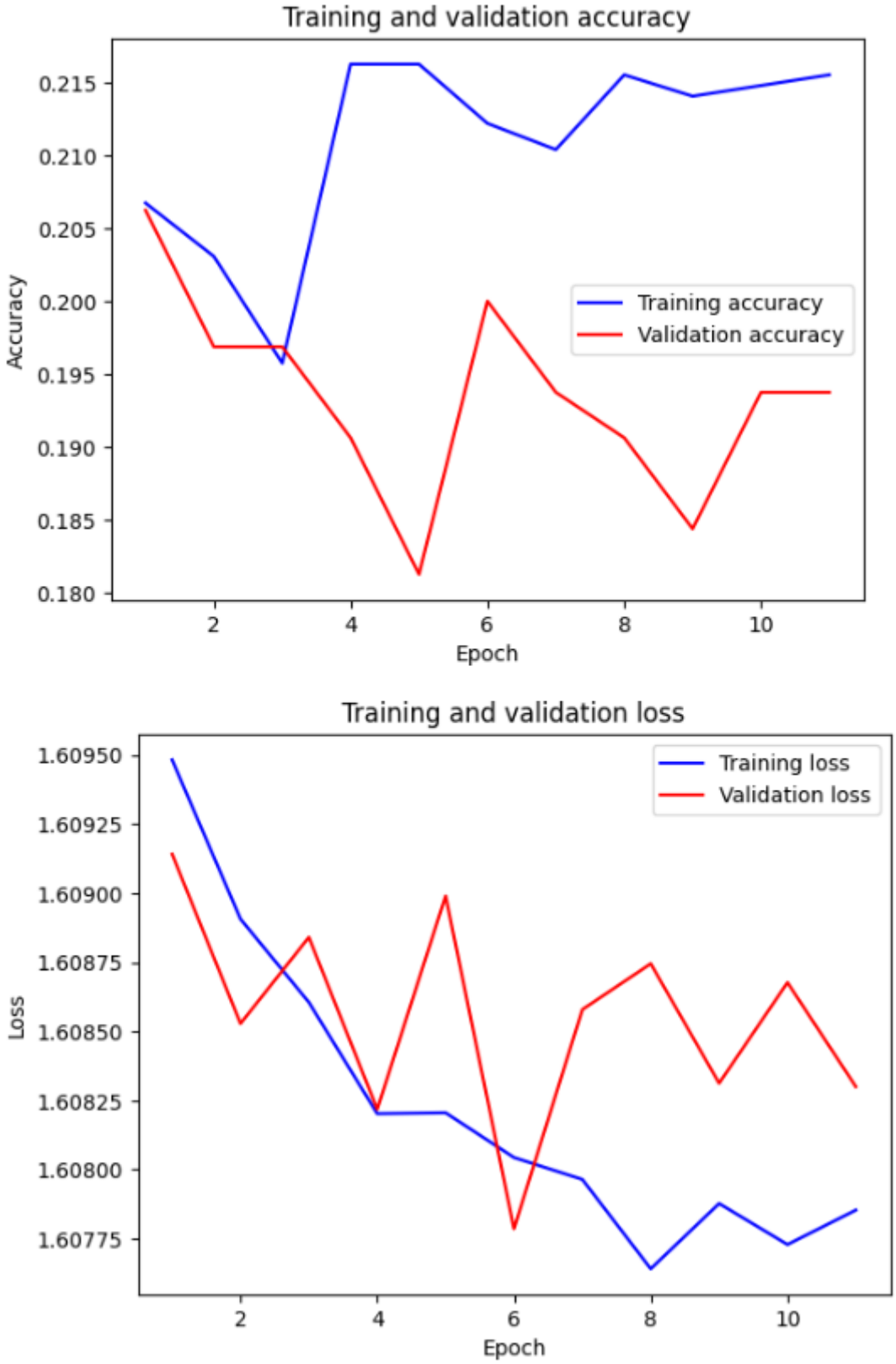


Figure 4.2.16 MobileNetV2 Without pretrained weight (apple types).

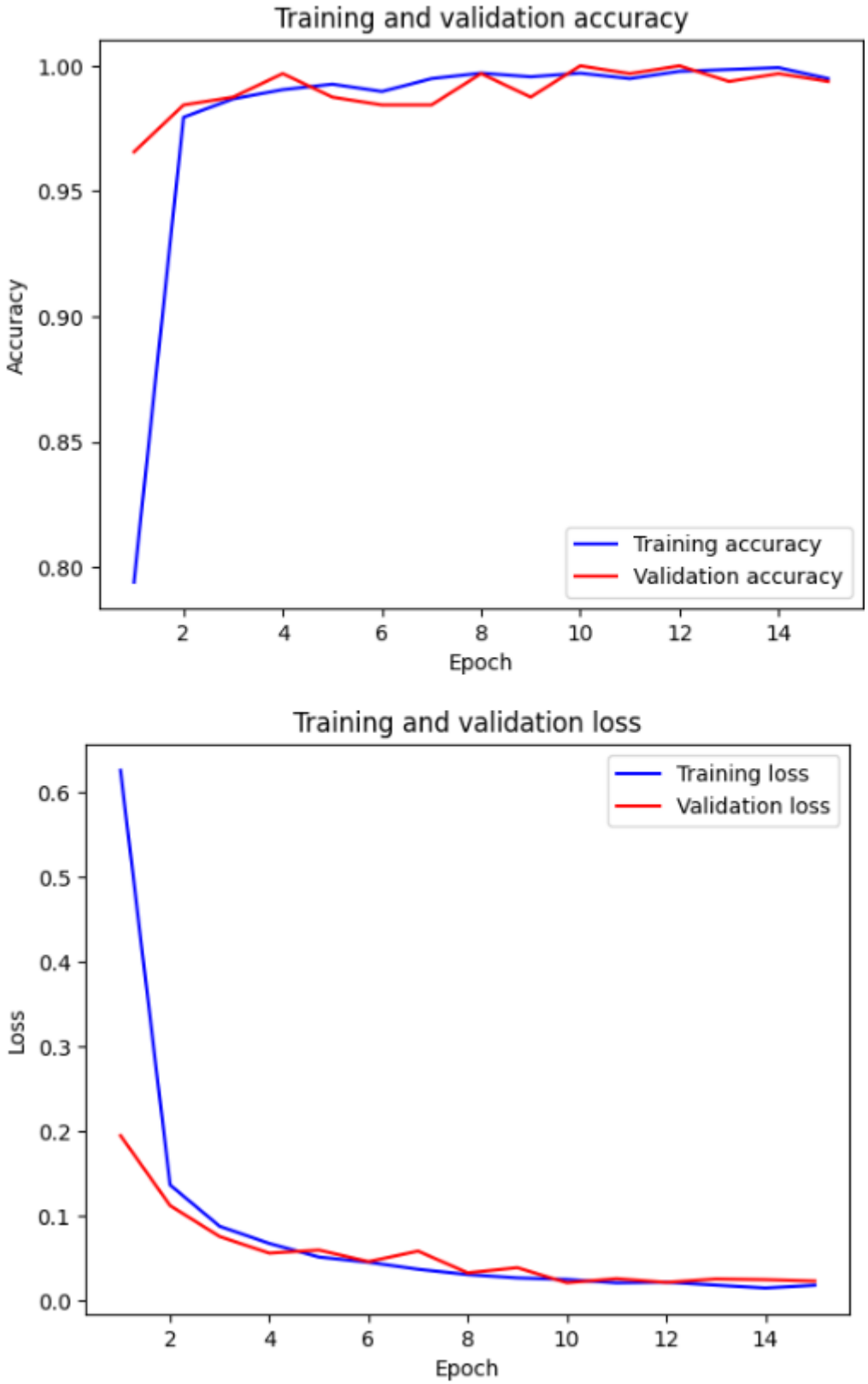


Figure 4.2.17 MobileNetV2 with pretrained weight (apple types).

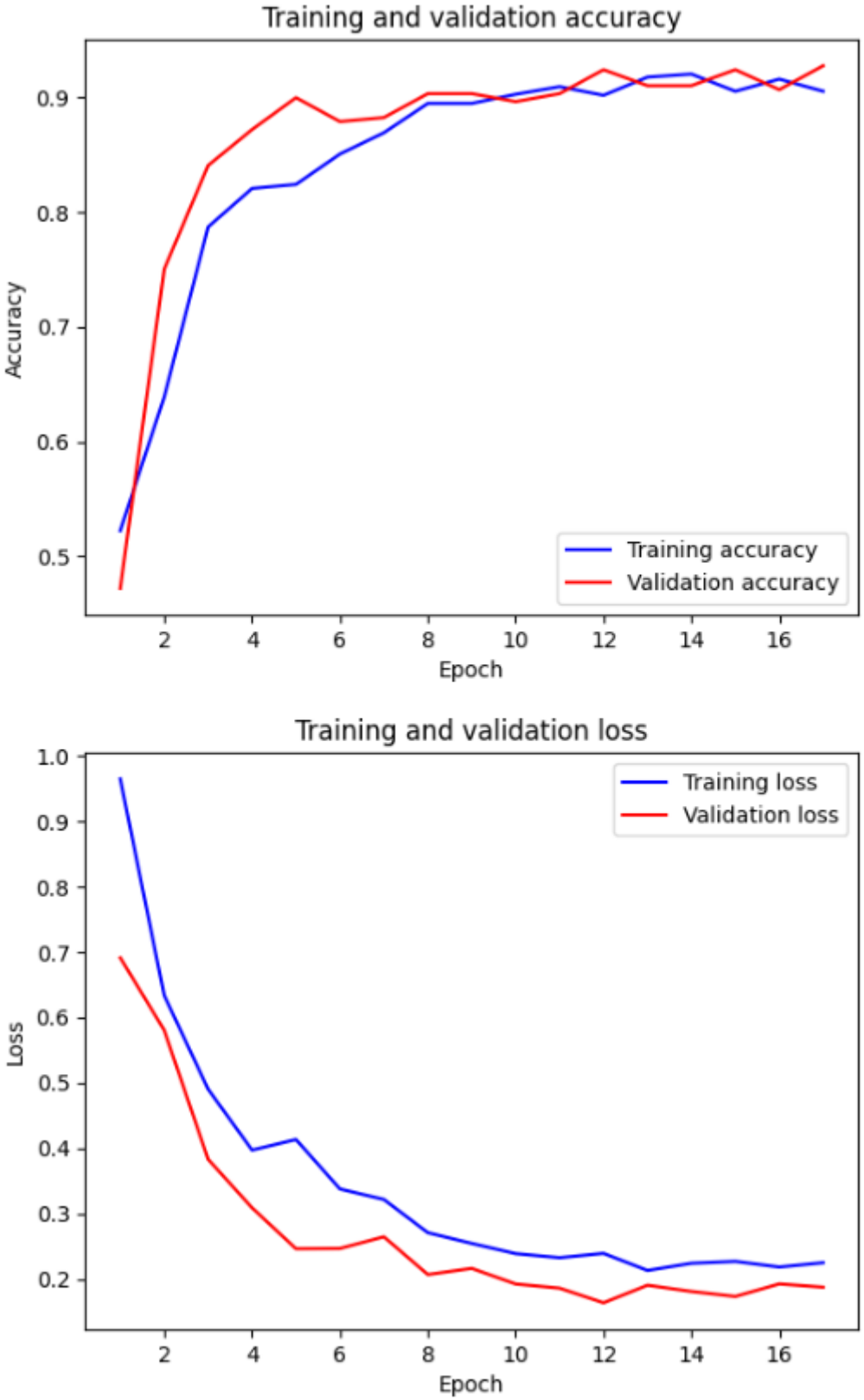


Figure 4.2.18 Self-defined custom CNN (apple conditions).

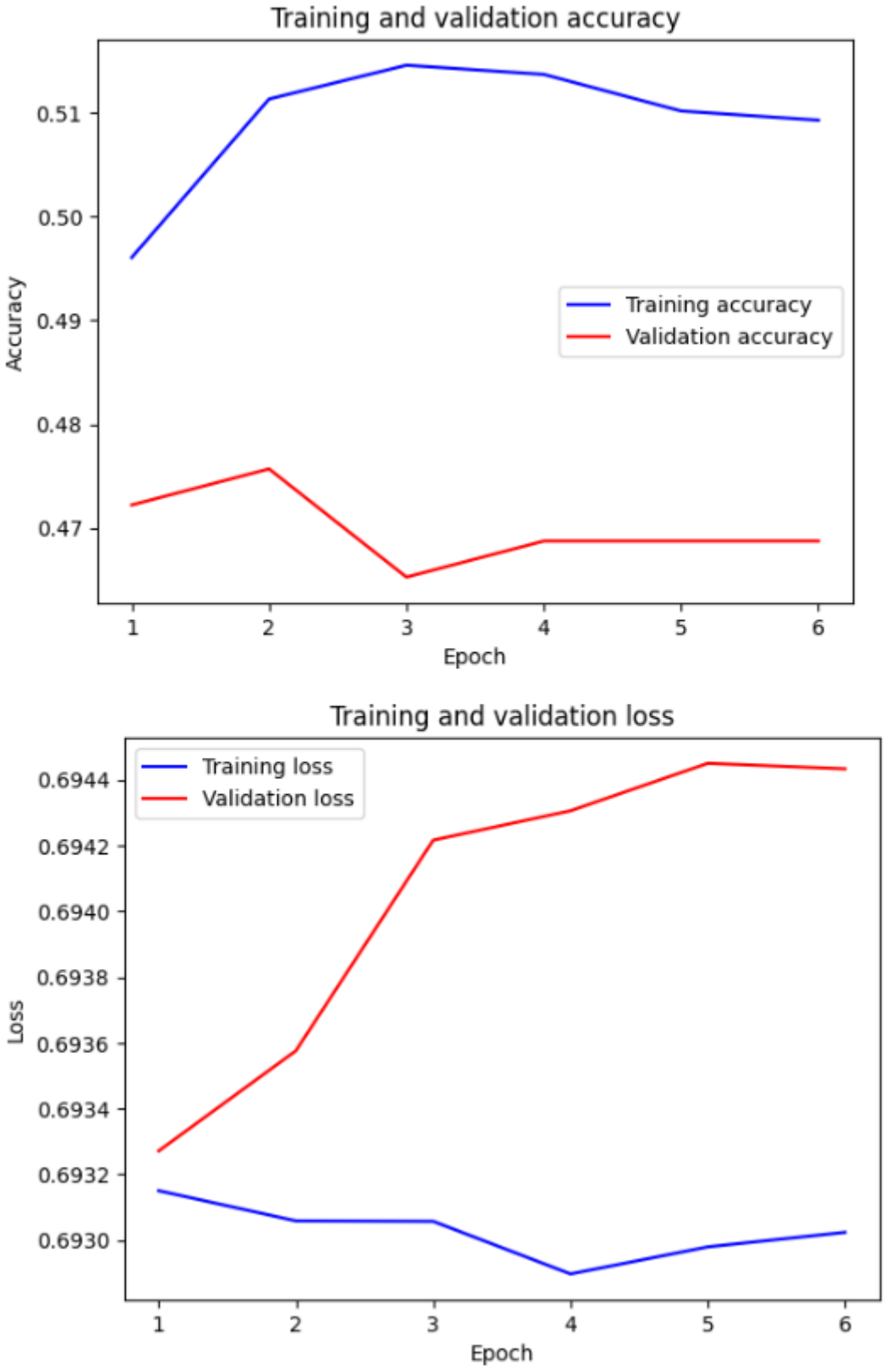


Figure 4.1.19 MobileNetV2 Without pretrained weight (apple conditions).

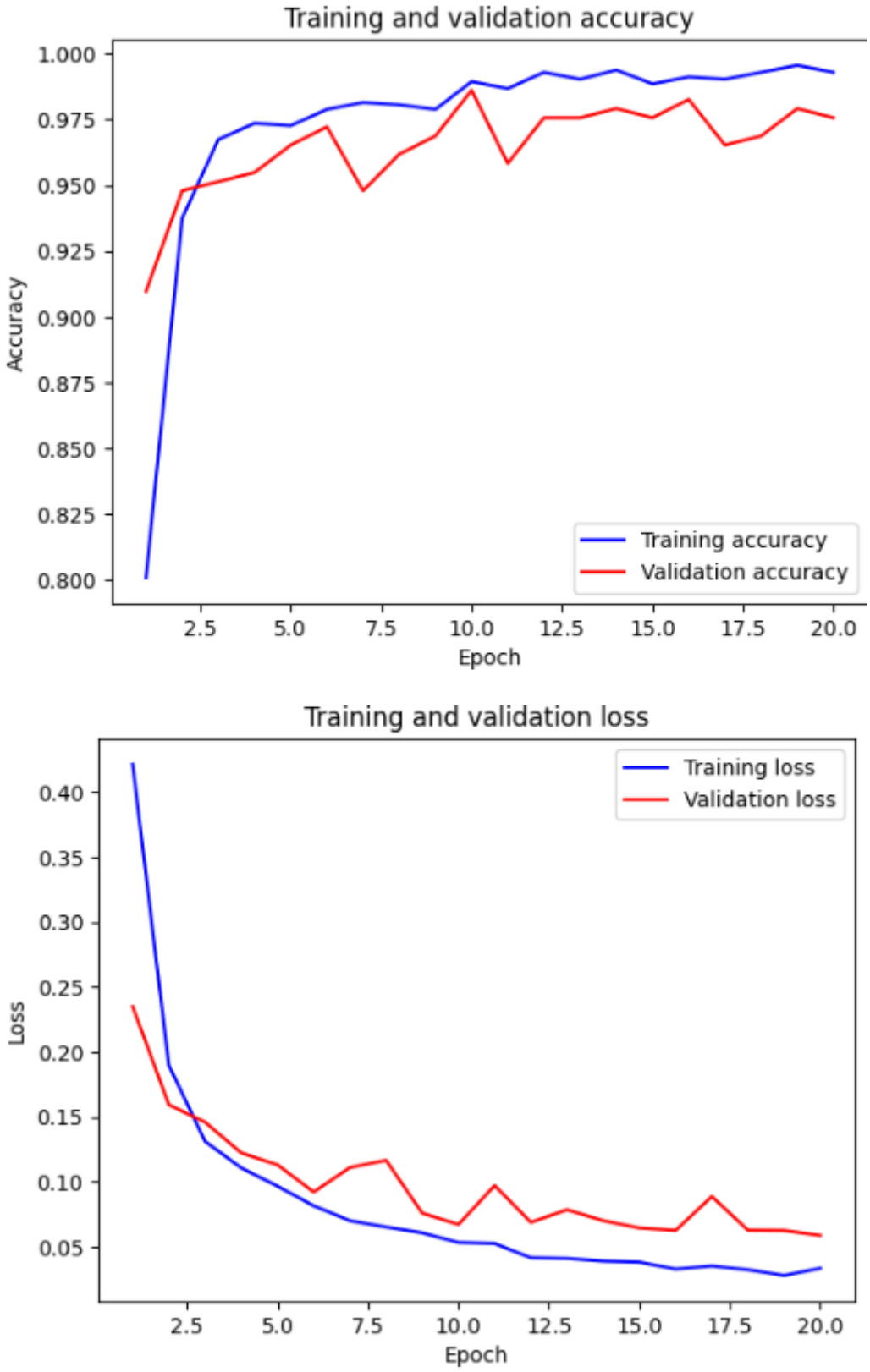


Figure 4.2.20 MobileNetV2 with pretrained weight (apple conditions).

4.2.9 Model Testing

In the final step of deep learning model development, the trained model is utilized to predict the labels of images from the test set. Fifteen images are randomly selected for this purpose, each image is displayed along with a title that includes its actual and predicted labels. As the visualization process is consistent across all models, the self-defined CNN model is chosen for demonstration. Figure 4.2.21 presents the predictions made by the custom CNN model. A detailed analysis of the model testing results, including accuracy and error rates, will be discussed in Chapter 6.

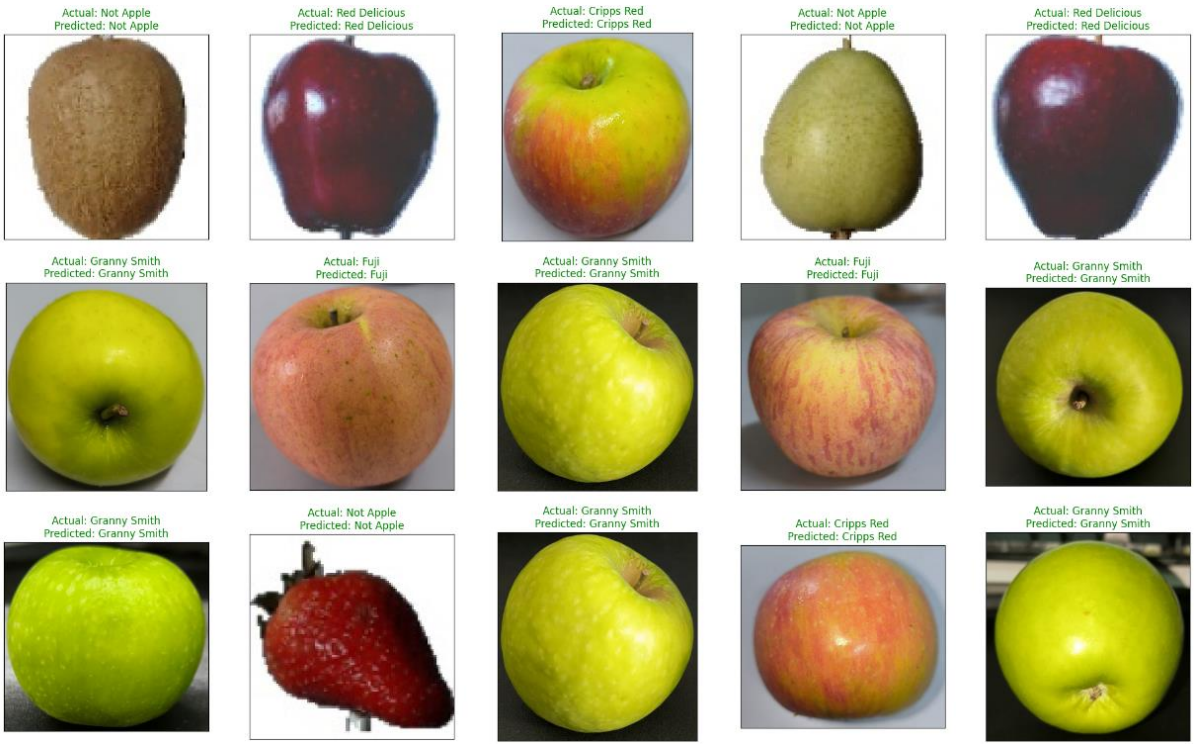


Figure 4.2.21 Custom CNN model predictions for apple types.

4.3 System Components Specifications (Overall Workflow)

The overall workflow of the system, as shown in Figure 4.1.2, is divided into three main phases, starting with development, deployment, and output. In the development phase, Visual Studio Code is used to design and structure the mobile application code, while Jupyter Notebook serves as the environment for developing and evaluating multiple deep learning models. The best model is selected and saved as a TensorFlow Lite file for the following deployment. This phase ensures both the mobile application and the deep learning model are ready for integration.

During the deployment phase, the trained TensorFlow Lite model is integrated into the complete mobile application code. This integration involves deploying the trained model to the mobile application and ensuring that the model is properly used for inference. The integration application code is then exported as an APK (Android Application Package) file. At this point, it is ready to be used and installed on Android devices.

Finally, in the output phase, the Flutter framework is utilized to build and install the final application on Android devices. The application is designed to classify apple types and conditions based on user-uploaded images or photos taken via the device's camera. By following this structured workflow, the system provides an effective solution for image classification tasks specific to apple types and conditions.

Chapter 5

System Implementation

5.1 Hardware Setup

The hardware setup for this project includes a desktop PC, a mobile device, and data storage devices. The desktop PC serves as the development machine, which is primarily used for designing and coding the mobile application, as well as training and evaluating the deep learning models. On the other hand, the mobile device functions as the testing platform to ensure the application operates correctly and efficiently on a real-world Android environment. Data storage such as local hard drive, is managed on the development machine where all datasets are stored locally. In addition, regular backups of the data are maintained on an external hard drive to safeguard against data loss and computer hardware failure.

5.2 Software Setup

This section provides screenshots along with descriptions to guide the software setup process. Further details on additional setup and configuration will be covered in Chapter 5.3 Setting and Configuration. This section focuses solely on demonstrating the download process.

5.2.1 Python

Python is the core programming language used for this project, particularly for developing and training the deep learning models. With Python's extensive library support such as TensorFlow and Keras, makes it an ideal choice for machine learning tasks. Moreover, libraries such as Pandas, Numpy, and Scikit-learn enables efficient data preprocessing and data evaluation in this project. First, open browser and go to website https://www.python.org/downloads/ to download and install Python as shown in Figure 5.2.1.

Figure 5.2.1 Python download website.

5.2.2 Jupyter Notebook

Jupyter Notebook is used as the deep learning model development environment due to its interactive and user-friendly interface, which allows for writing and running Python code in a cell-based format. Several reasons make Jupyter Notebook an ideal choice for this project, including its straightforward setup, no additional dependencies required, and cross-platform compatibility. To install Jupyter Notebook, type “pip install notebook” in the command prompt. Once the installation is complete, launch it by typing “jupyter notebook”. This action will display results similar to Figure 5.2.2. After successfully opening, Jupyter Notebook's homepage will be accessible at the URL http://localhost:8888/tree, as shown in Figure 5.2.3.

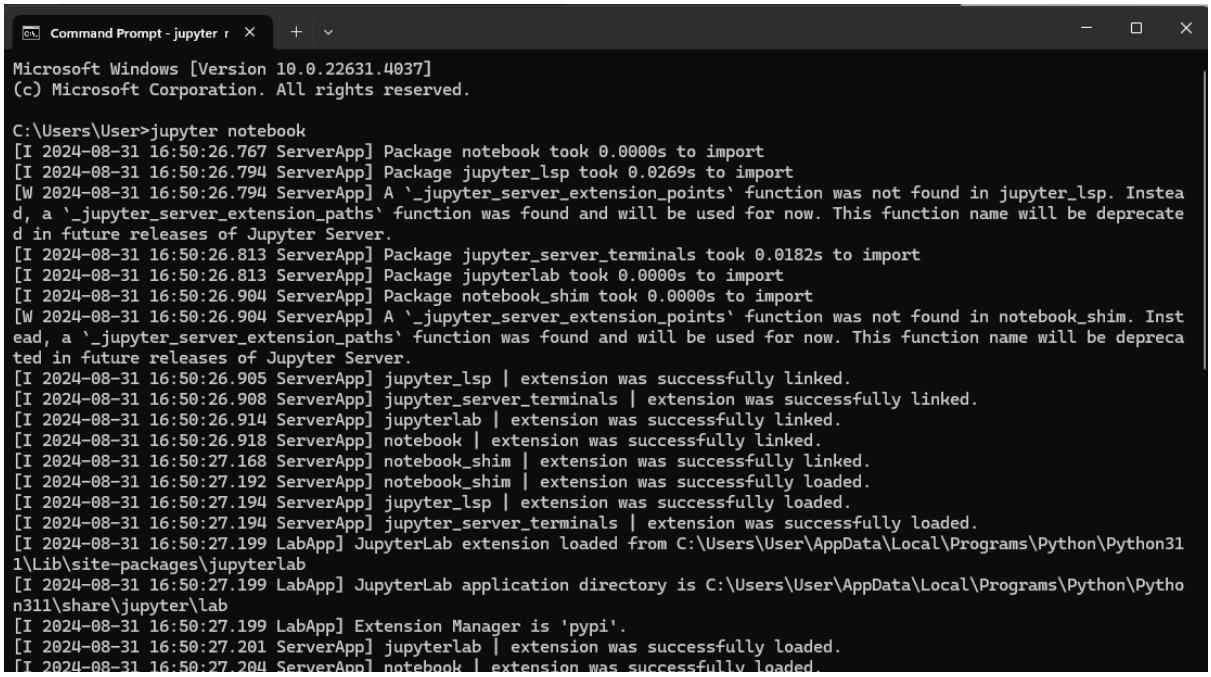


Figure 5.2.2 Open Jupyter Notebook in command prompt.

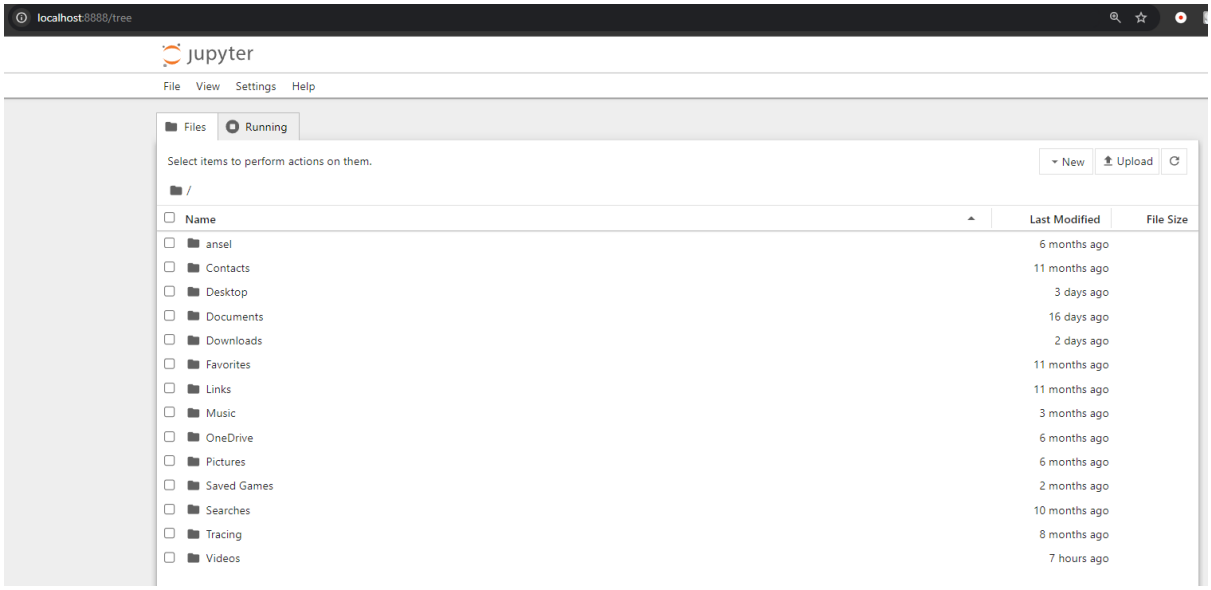


Figure 5.2.3 Jupyter Notebook's homepage.

5.2.3 Visual Studio Code

Visual Studio Code is used for coding the mobile application. It provides flexible Integrated Development Environment (IDE) with extensions that support Dart and Flutter development. While Visual Studio Code is primarily used for writing and managing the codebase, external features like debugging tools enhance the development process. In this case, Android Studio

emulator is utilized for testing, debugging, and running the application in order to ensure that the application works correctly in a simulated Android environment.

Open the browser and go to website https://code.visualstudio.com/download to download Visual Studio Code as shown in Figure 5.2.4.

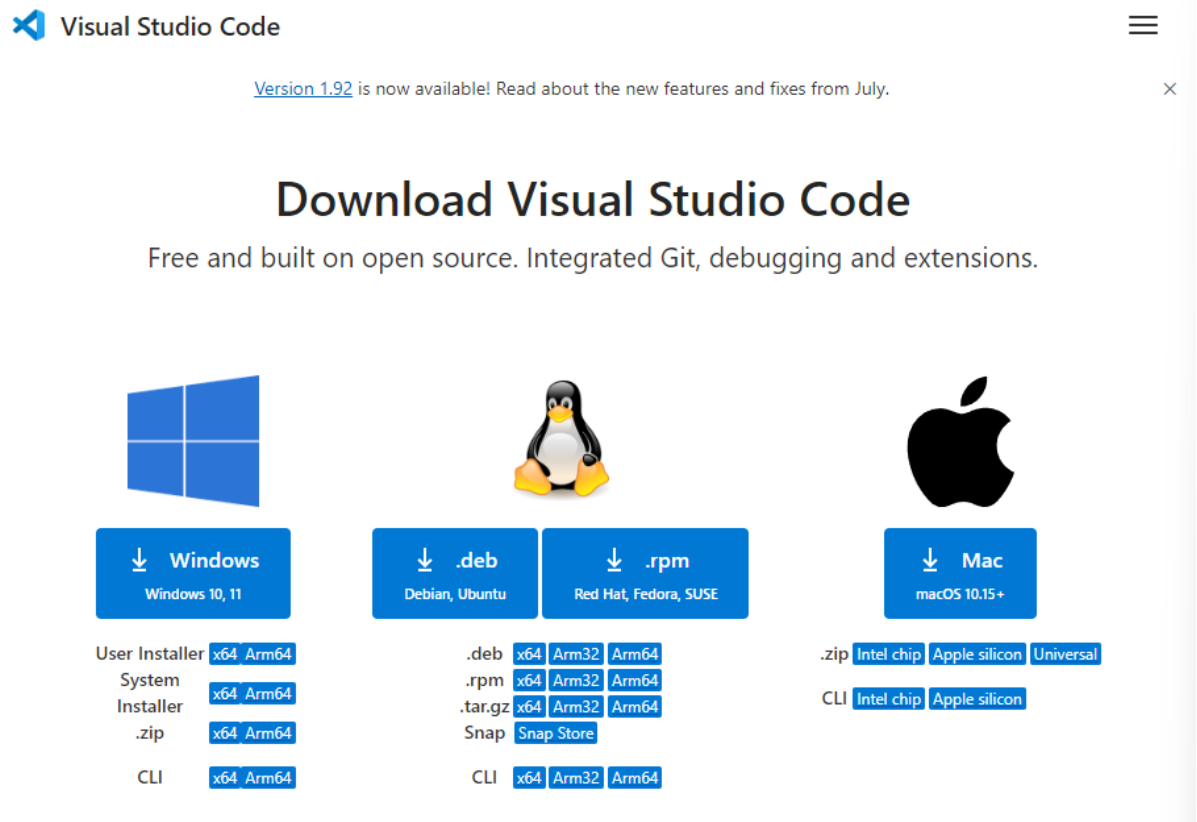


Figure 5.2.4 Visual Studio Code download page.

5.2.4 Flutter

Flutter is an open-source UI software development toolkit, it is the chosen framework for building the mobile application. It allows for the creation of natively compiled applications for mobile from a single codebase. In this project, Flutter is paired with Dart to develop the user interface and integration the trained model for classification of apple types and conditions.

Firstly, open the browser and navigate to the website https://docs.flutter.dev/get-started/install/windows/web. Scroll down until the "Install the Flutter SDK" section is displayed then click the "flutter_windows_3.24.1-stable.zip" to begin the download. Figure 5.2.5 provides the example of the download page. Once the Flutter SDK installation is

completed, open Visual Studio. In the left navigation bar, search for Flutter and Dart in the Extensions section, and download both extensions. Figure 5.2.6 demonstrates the example of both extensions.

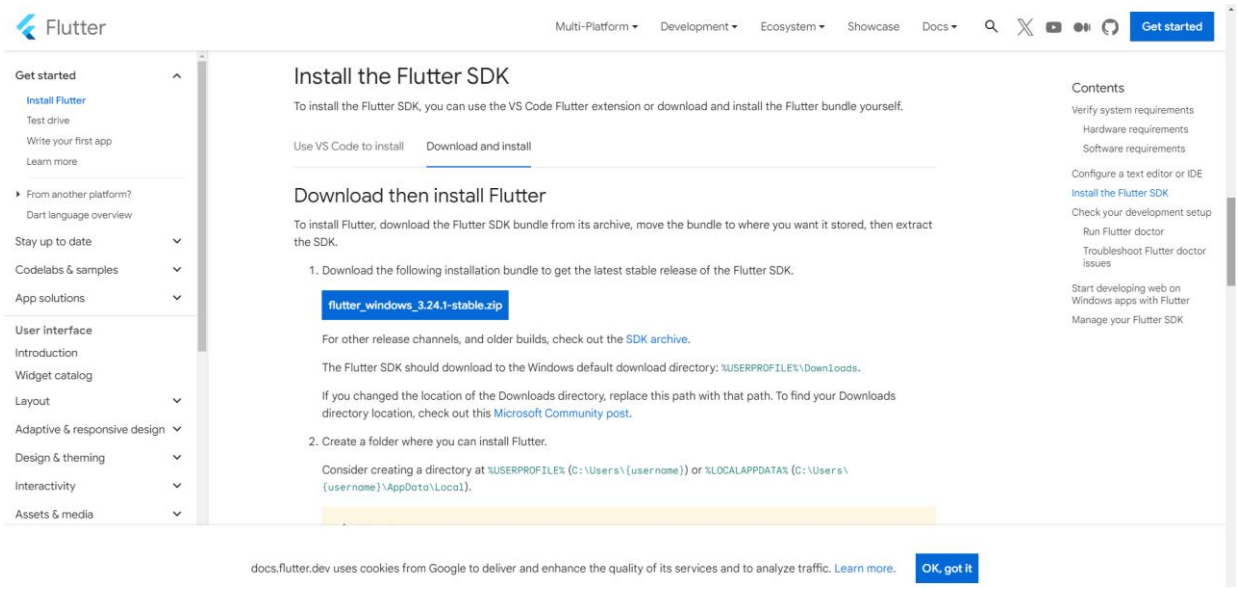


Figure 5.2.5 Flutter download page.

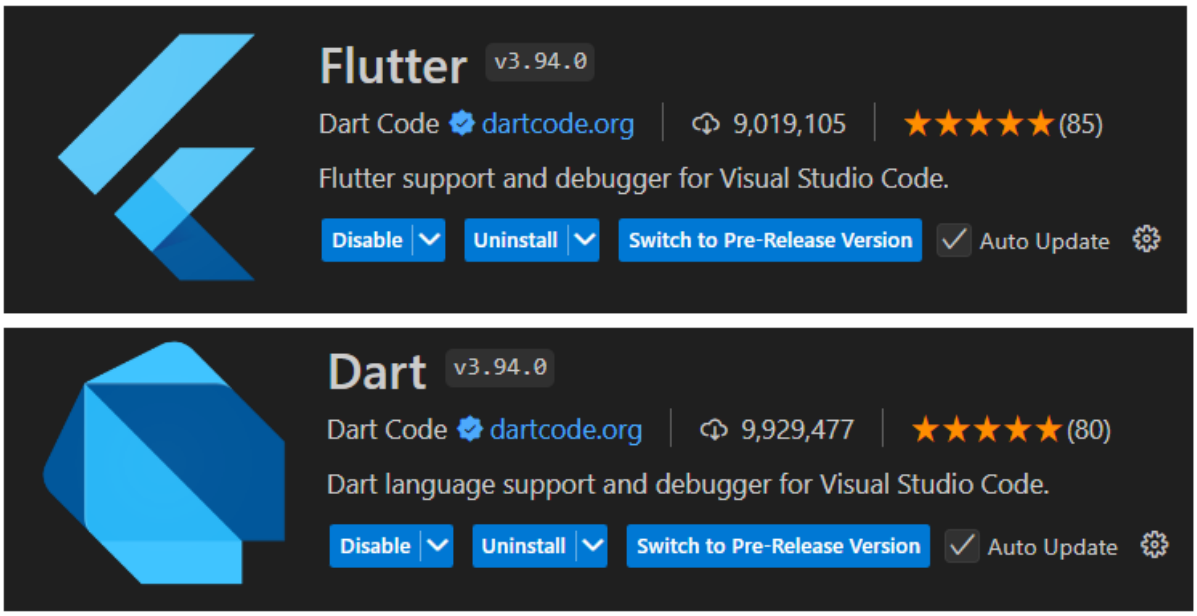


Figure 5.2.6 Flutter and Dart extension in Visual Studio Code.

5.2.5 Android Studio

Android Studio is used to compile, build and test the Flutter application on Android devices. Although the software includes comprehensive tools for Android app development, such as a

code editor, debugging tools, and the ability to generate APK files. However, it primarily serves as mobile device emulator in this project. This emulator allows for the simulation of Android devices to test the application's functionality and user interface. Notably, the emulator dynamically reflects real-time changes to the UI when code is edited, providing immediate feedback on how those changes affect the application's appearance and behaviour. This makes the development process significantly convenient and efficient.

The download method is similar to software discussed in the previous subsections (5.2.1, 5.2.3, and 5.2.4). Open the browser, go to website https://developer.android.com/studio, and click the download button to start the download. Figure 5.2.7 displays the download page of Android Studio.

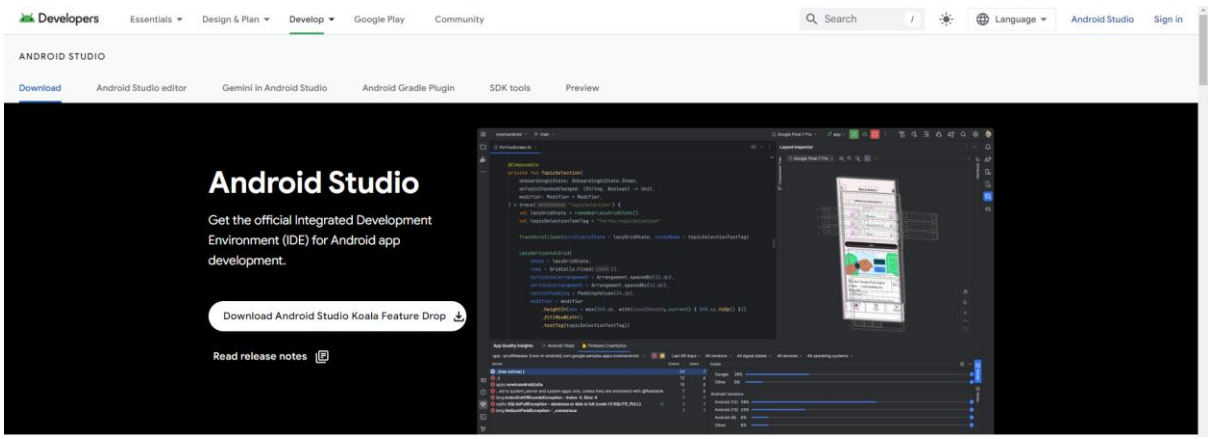


Figure 5.2.7 Android Studio download page.

5.3 Setting and Configuration

Several settings and configurations are required for Python, Flutter, and Android Studio before implementing the system. For example, Python, Flutter, and Android Studio. These components need to be correctly set up and configured to ensure that it is working as expected. Each of these components setting and configuration will be discussed in detail in separate subsections.

5.3.1 Python

The environment path should be configured to ensure that Python commands can be executed from any command prompt. This requires adding the installation directory to the system's PATH variable, allowing the system to recognize Python commands globally. The correct path

should look like “C:\...\...\AppData\Local\Programs\Python\Python311”. Figure 5.3.1 shows the example of configured environment variables. Next, open command prompt and type “python” to verify if it is installed and configured successfully. It will show the result similar to Figure 5.3.2.

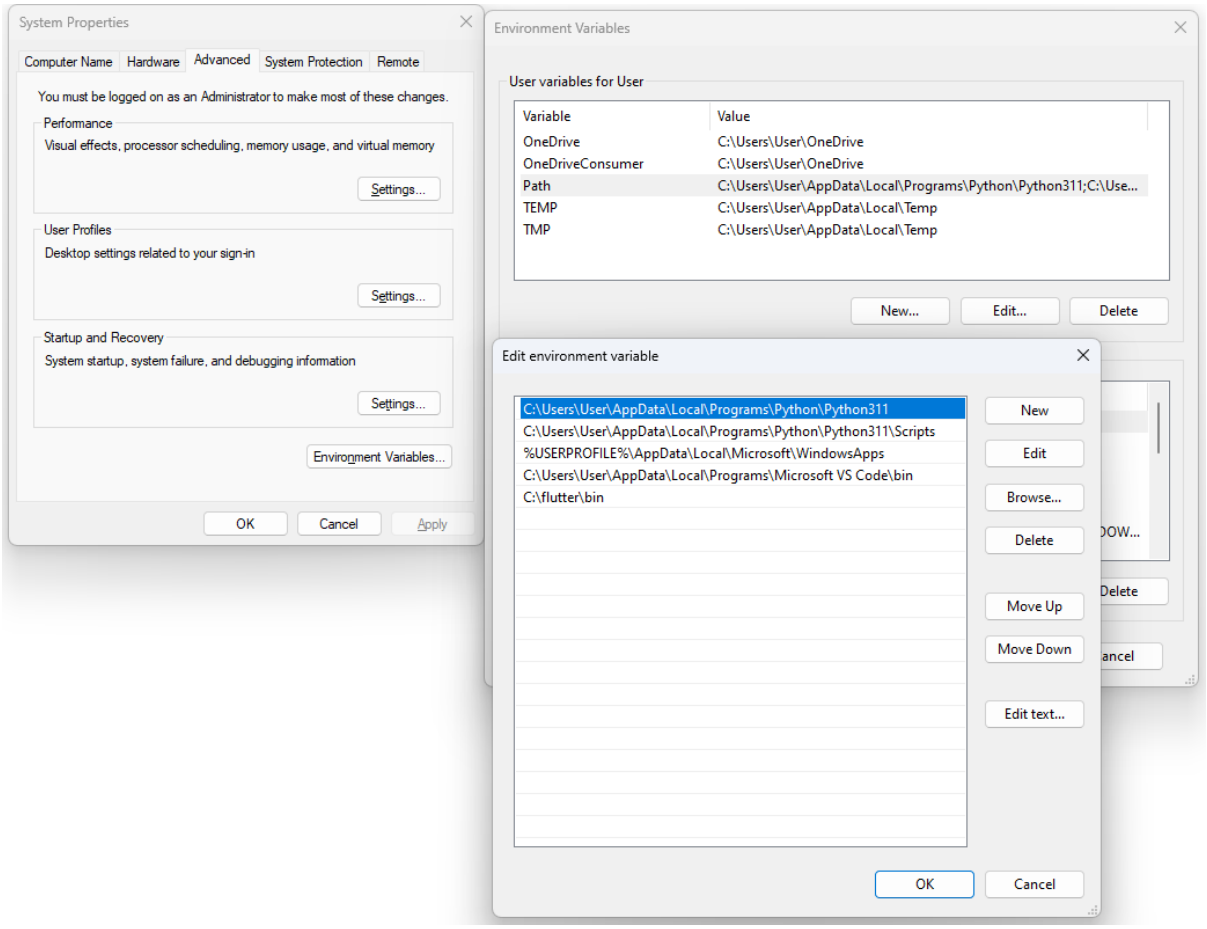


Figure 5.3.1 Configured environment variables for Python.

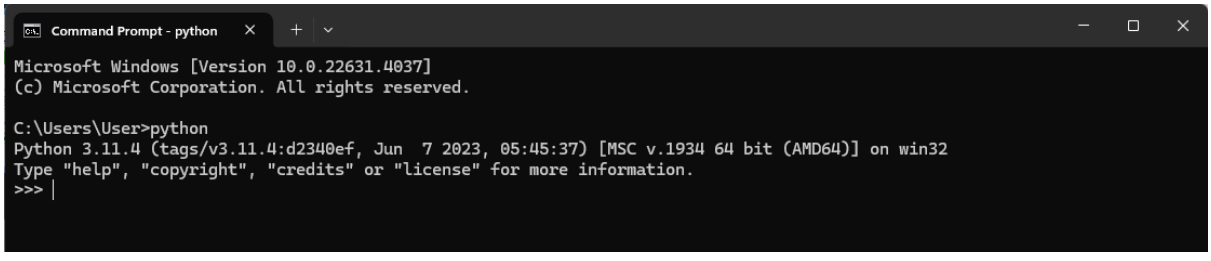


Figure 5.3.2 Python download verification in command prompt.

5.3.2 Flutter

As the downloaded Flutter SDK file shown above in Figure 5.2.5 is in a ZIP format, it must be extracted. After extracting, locate a folder named “flutter” and move it to the C drive. This placement ensure that the Flutter SDK is accessible for system configurations. Repeat the environment path configuration as shown in Figure 5.3.1. The new directory path to be added is “C:\flutter\bin”. By doing this, Flutter commands can be executed from any command prompt on the system. To verify the installation and configuration, open a command prompt and type “flutter doctor”. If Flutter is installed and configured correctly, this command will display a summary of the Flutter environment setup similar to the output shown in Figure 5.3.3. In addition, type both “flutter --version” or “flutter can also verify if Flutter is installed and configured correctly”.

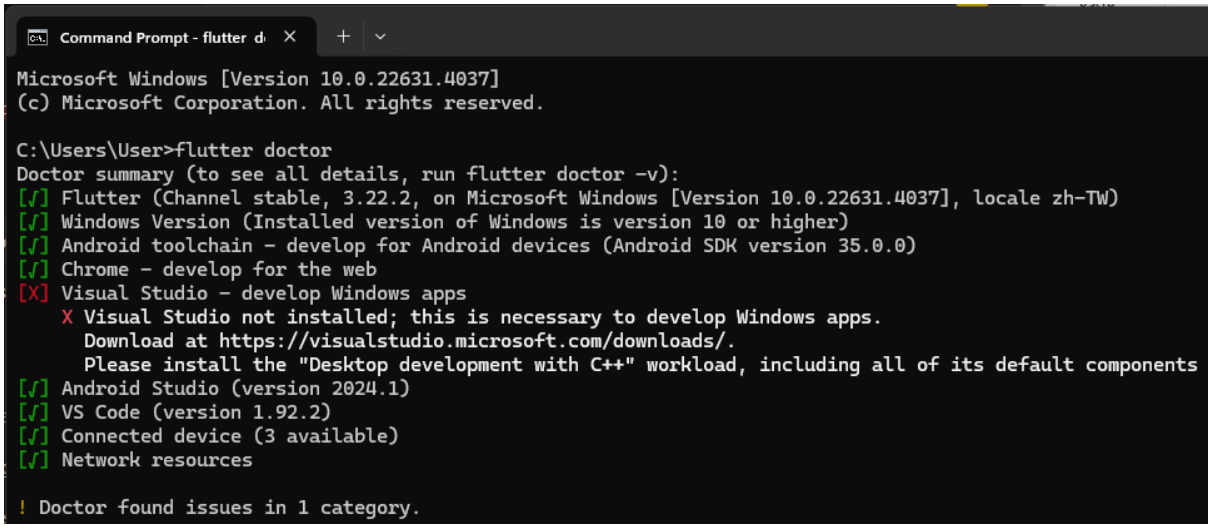


Figure 5.3.3 Example of configured flutter environment.

5.3.3 Android Studio

For setting up Android Studio to use the emulator, the following configuration steps are required. Starting by launching Android Studio. On the welcome screen, click on the “More Actions” located below the "New Flutter Project," "New Project," and "Open" buttons and select “SDK Manager” from the dropdown menu. In the SDK Manager, navigate to the "Languages & Frameworks" section and select "Android SDK." From there, click on the "SDK Tools" tab. In this tab, check the option for "Android SDK Command-line Tools (latest)." After selecting this option, click the "Apply" button to begin the installation process. Figure 5.3.4 shows the example of Android SDK Manager settings.

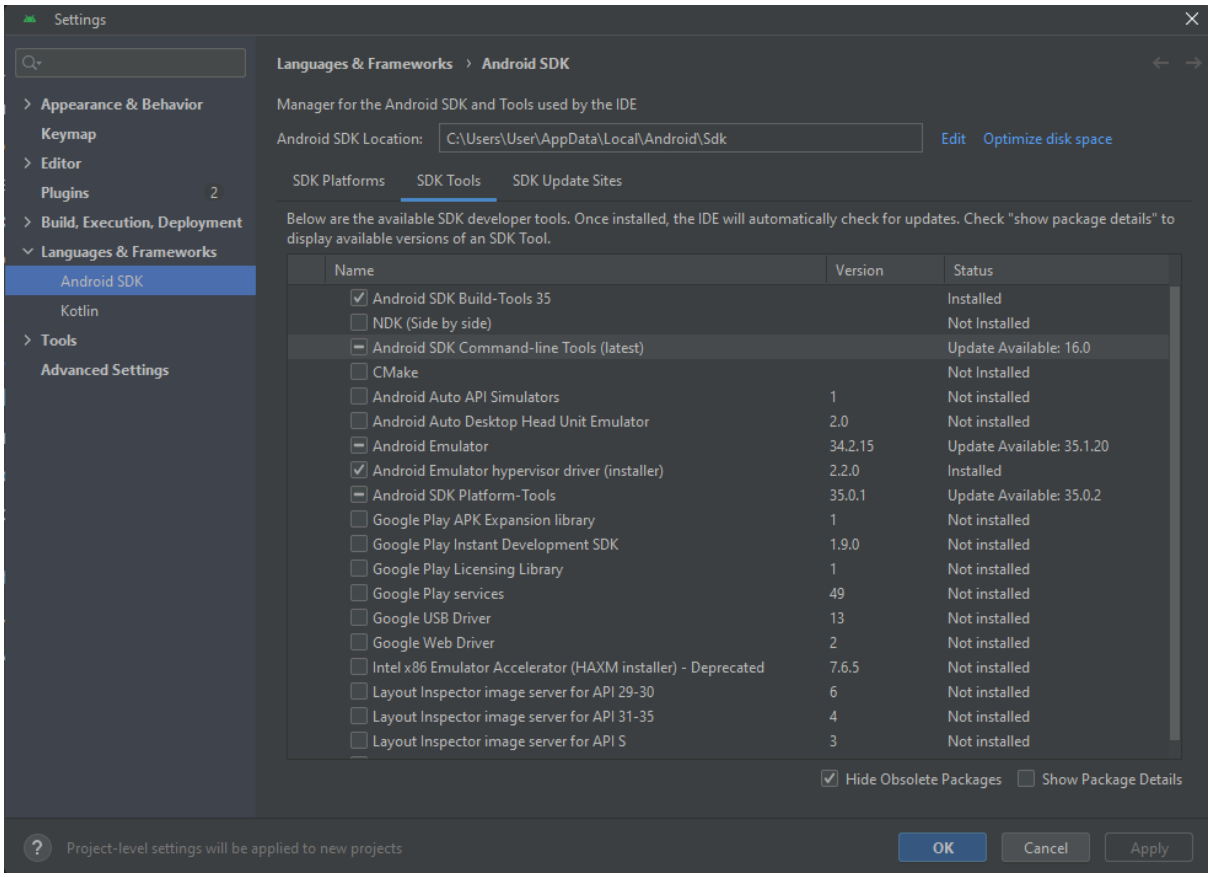


Figure 5.3.4 Android Studio settings window.

Another important component to setup is the Android toolchain. To complete the setup for Android Studio and ensure that the Android toolchain is correctly configured, it is necessary to use the command prompt. Type “flutter doctor” in the command prompt. If there is a “[!]” symbol next to the Android toolchain as illustrated in Figure 5.3.5, it indicates that additional configuration is needed. To resolve this, type “flutter doctor –android-licenses” to proceed the configuration. Follow the prompts by typing “y” to accept all the licenses. If it is correctly configured, the output should resemble what is shown in Figure 5.3.3.

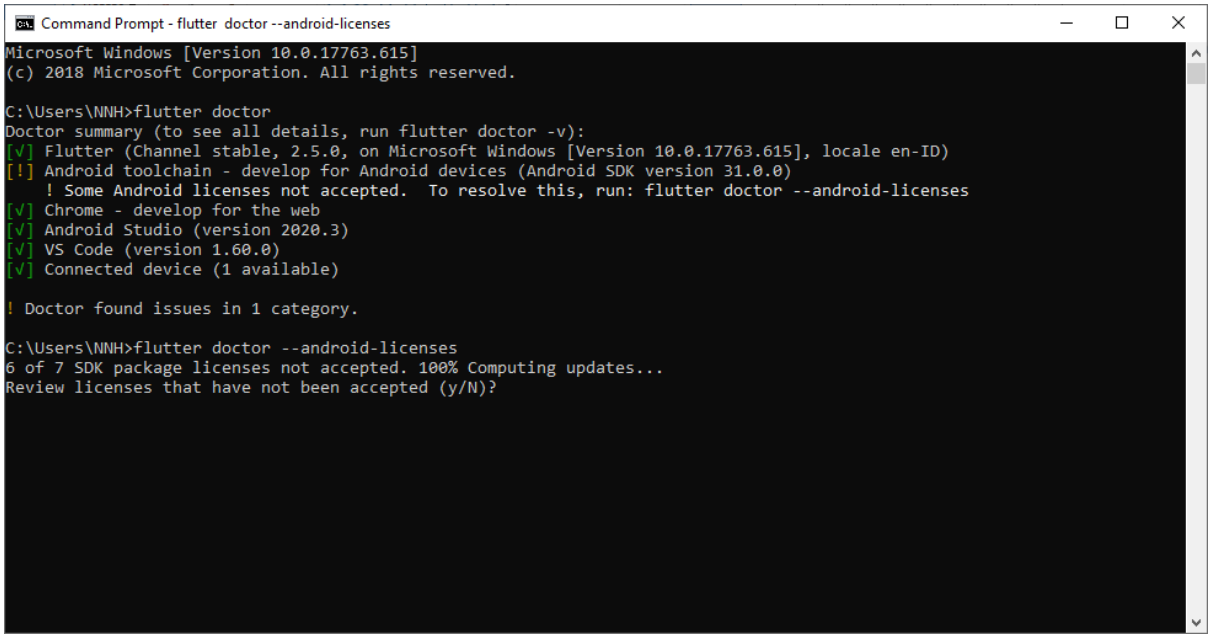


Figure 5.3.5 Configuration of Android toolchain in command prompt.

For the final step of configuring Android Studio, click on the "More Actions" button again, then select "Virtual Device Manager" from the dropdown menu. Click "Create Virtual Device," choose a device definition (such as Pixel XL), and then select a system image (for example, R, Android 11). Wait for the download to complete. Once the download is finished, click "Next," then "Finish." After this setup, the emulator will be available and can be launched anytime by selecting "Virtual Device Manager." Figure 5.3.6 shows the available virtual device in device manager.

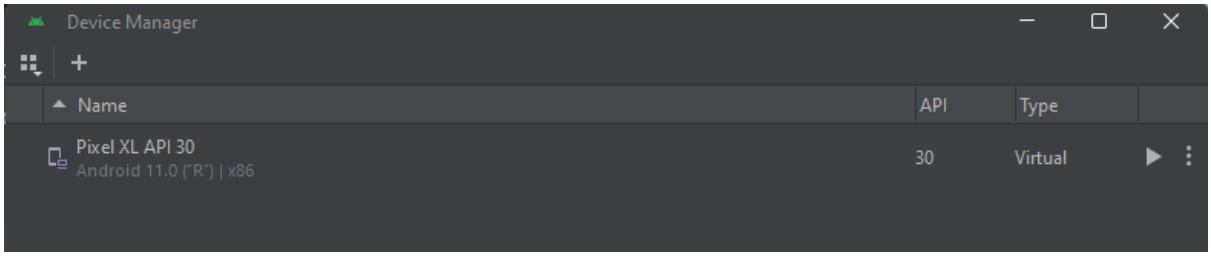


Figure 5.3.6 Virtual device manager of Android Studio.

5.3.4 Visual Studio Code

For setting up Visual Studio Code, start by clicking "Open Folder" on the welcome screen and select or create a new empty folder for the project. Once the folder is selected, open the terminal within Visual Studio Code and enter the command “flutter create [project name]” as shown in Figure 5.3.7. This command will generate the necessary files for your Flutter project. After the

project is created, the code is ready to be written in the “main.dart” file located within the “lib” folder.

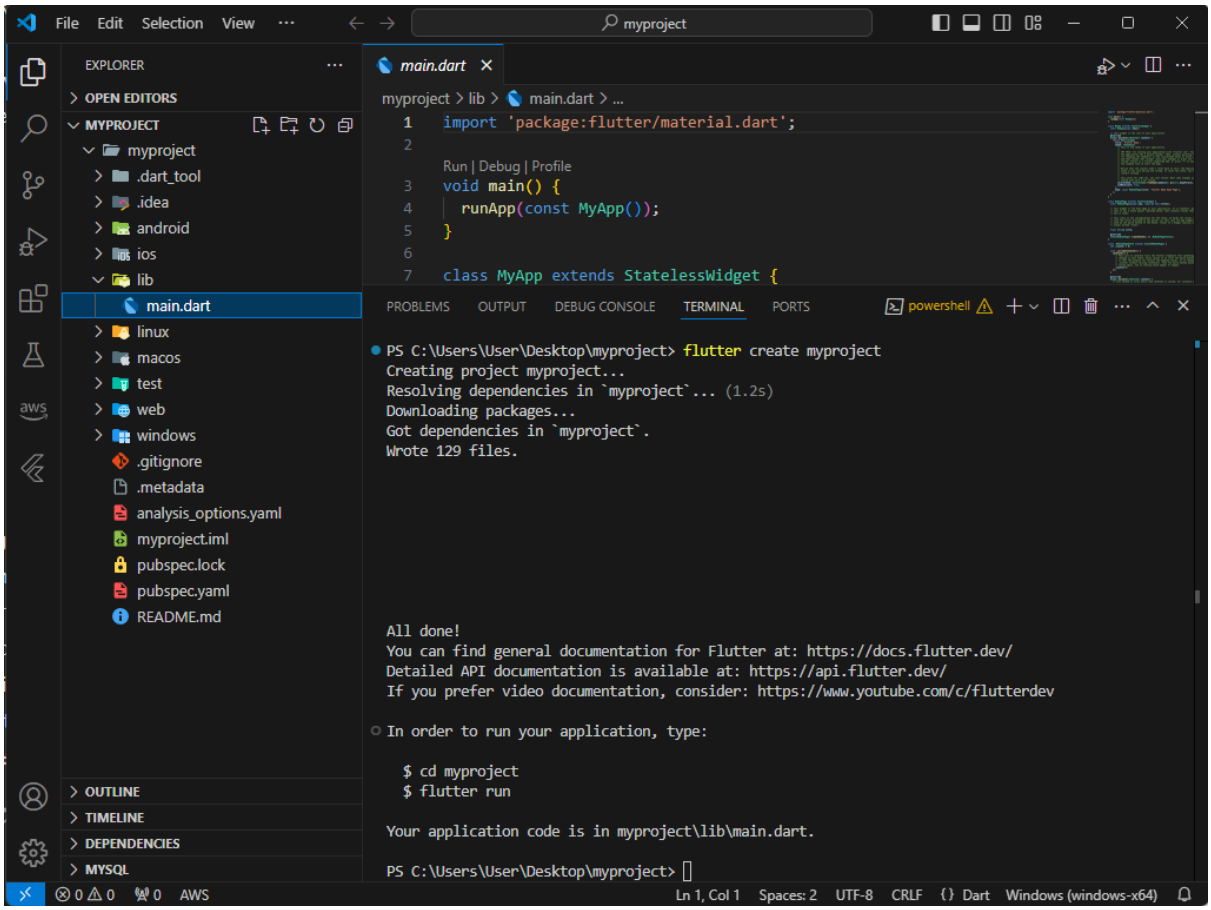


Figure 5.3.7 Example of new generated project.

To run the code on the emulator, click on “Windows {windows-x64}” located at the bottom right as shown in Figure 5.3.7. A dropdown menu will appear with a search bar at the top, then select the mobile emulator from this menu as illustrated in Figure 5.3.8. Figure 5.3.9 displays the launched emulator.

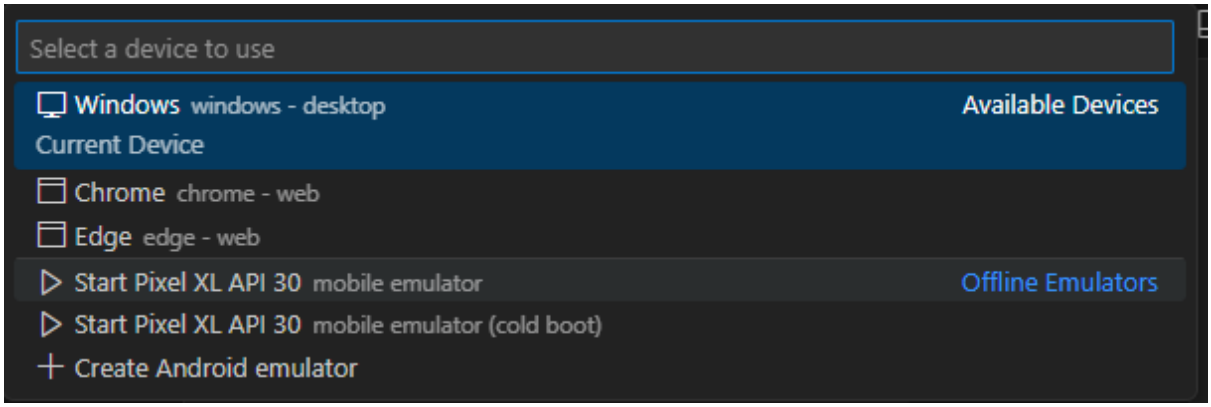


Figure 5.3.8 Drop down menu of debugging devices.

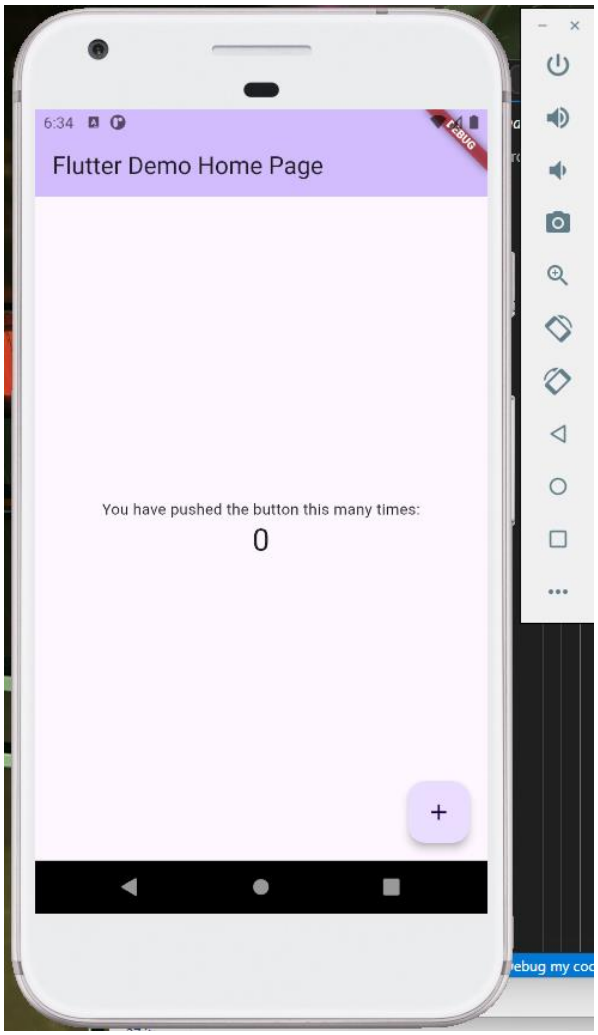


Figure 5.3.9 Launched mobile emulator.

The final setup involves creating an "assets" folder within the project folder to store the necessary files for development, such TFLite model files, images, and text files (labels). Figure 5.3.10 demonstrates the assets folder. After creating this folder, open the "pubspec.yaml" file

to import the required libraries for the project. After that, set the assets section in the "pubspec.yaml" to include "- assets/", which ensures that all files within the "assets" folder are accessible to the application. Figure 5.3.11 provides an example of configured "pubspec.yaml".

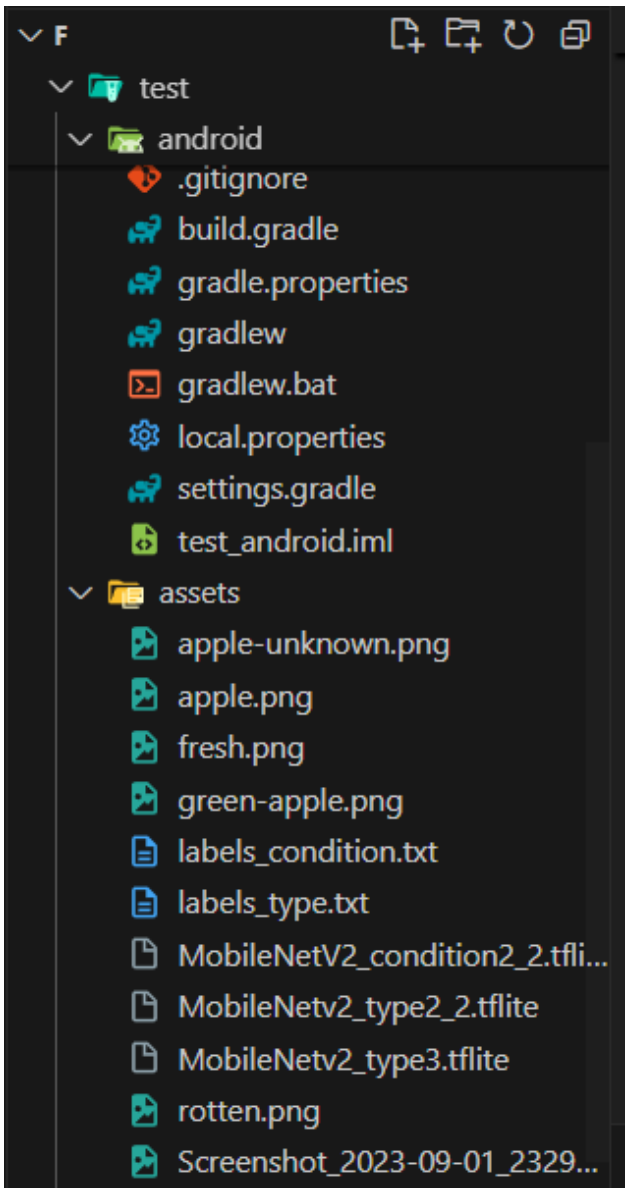


Figure 5.3.10 Assets folder.

```yaml
dependencies:
  flutter:
    sdk: flutter
  image: ^3.0.1
  image_picker: ^0.8.9
  flutter_tflite: ^1.0.1
```

```yaml
# To add assets to your application, add an assets section, like this:
assets:
  # - assets/MobileNetV2_type2_2.tflite
  # - assets/labels.txt
  - assets/
```

Figure 5.3.11 Configured .yaml file.

5.4 System Operation

The mobile application starts with an interface titled “Apple Types & Condition Classification” displayed at the top in a bold, white font against a blue background. At the center of the screen, a large grey box with a border is shown, containing the text "No image selected.". Tapping on this grey box triggers a menu at the bottom of the screen with two options, "Pick from Gallery" and "Take a Picture." The first option allows the user to select an image from the device’s gallery, while the second option opens the device’s camera for capturing a new image.

Once an image is selected or captured, it replaces the placeholder text and is displayed within the grey box. The selected image is shown with proper fitting and alignment. Below this grey box, there are two buttons, "Remove Image" and "Classify.". The "Remove Image" button clears the selected image, reverting the grey box to its initial state with the "No image selected" placeholder. A small notification appears at the bottom of the screen, displaying "Image removed." If the user attempts to classify without selecting an image first, a notification will appear, stating “Please select an image first” to notify the user.

After selecting an image, pressing the "Classify" button displays the classification results in a card format. The first card shows the name of the apple type (e.g., Cripps Red) along with a small apple icon. A second card appears below the first one, indicating the condition of the apple (e.g., fresh or rotten) with corresponding text and an icon. The card's background color changes based on the classification results, green for fresh, red for rotten, and yellow for Not Apple. The icon also changes according to the apple type, a red apple icon for Cripps Red, Fuji, or Red Delicious and a green apple icon for Granny Smith. At the end of the process, the user

can remove the image and start over by selecting a new image or taking another picture. Figure 5.4.1 to 5.4.8 presents the screenshots of system operation in mobile application.

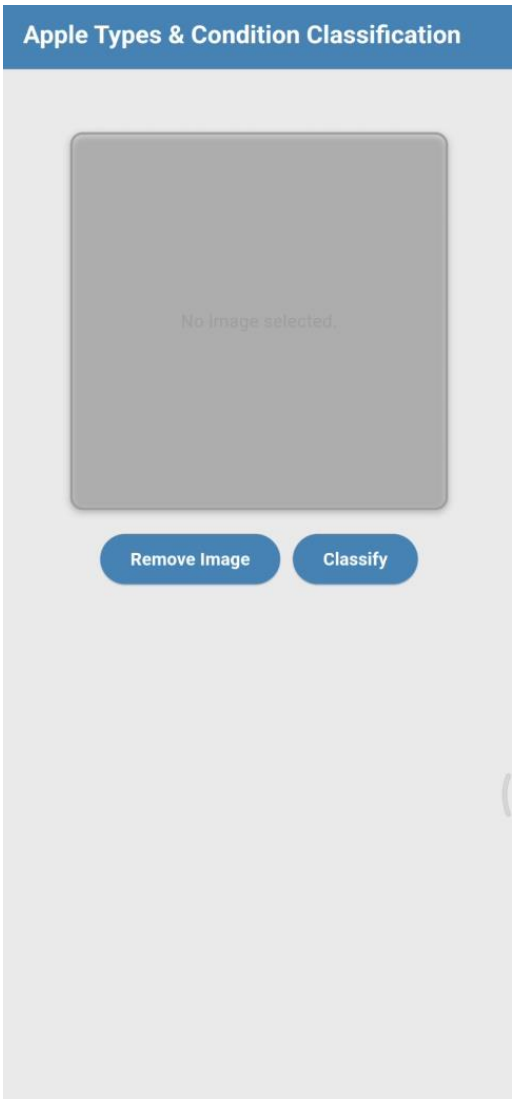


Figure 5.4.1 Mobile application UI.

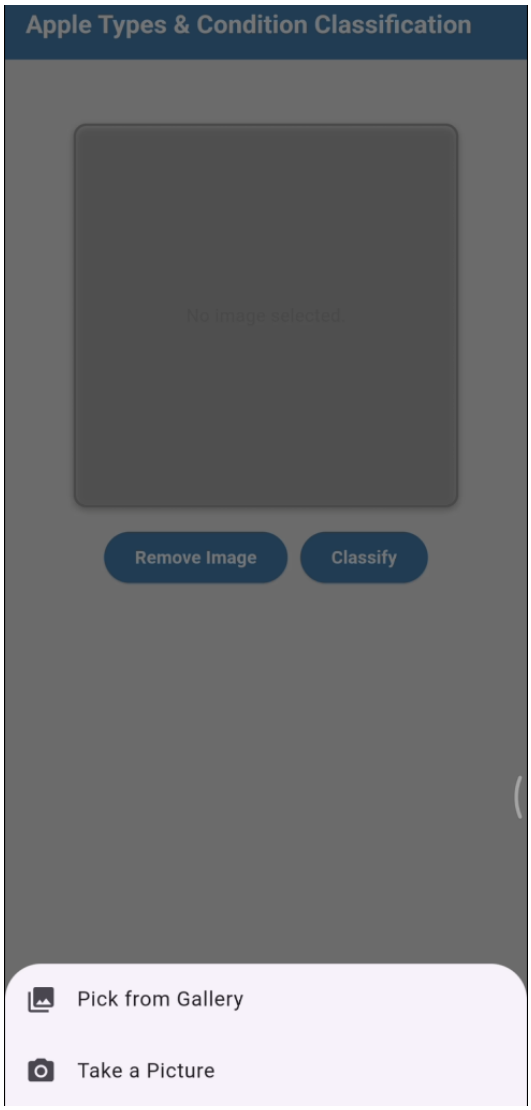


Figure 5.4.2 Bottom sheets with options.

Figure 5.4.3 Select image from gallery.

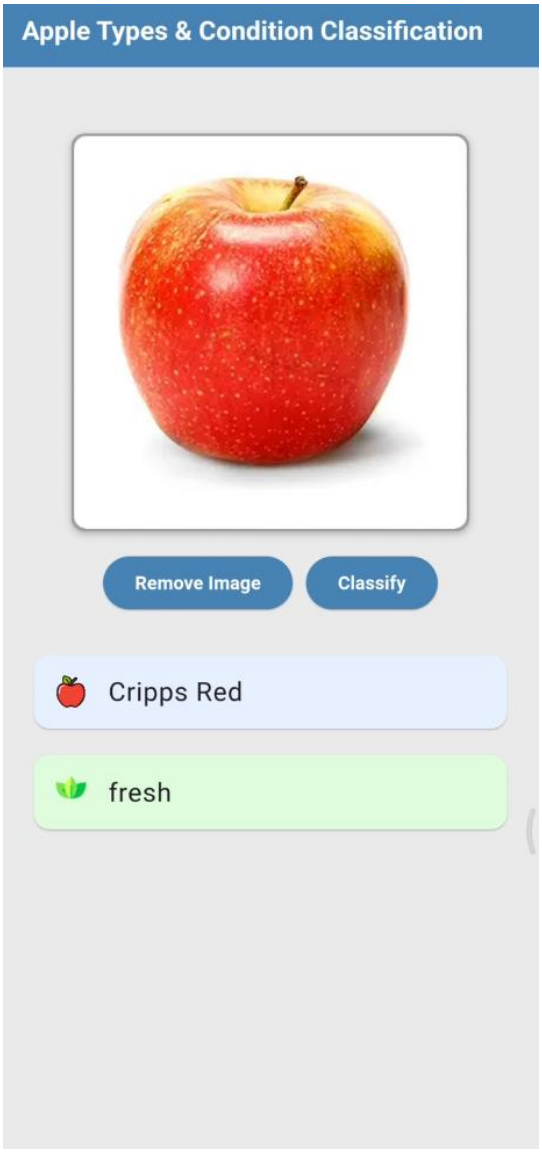


Figure 5.4.4 Classification result.

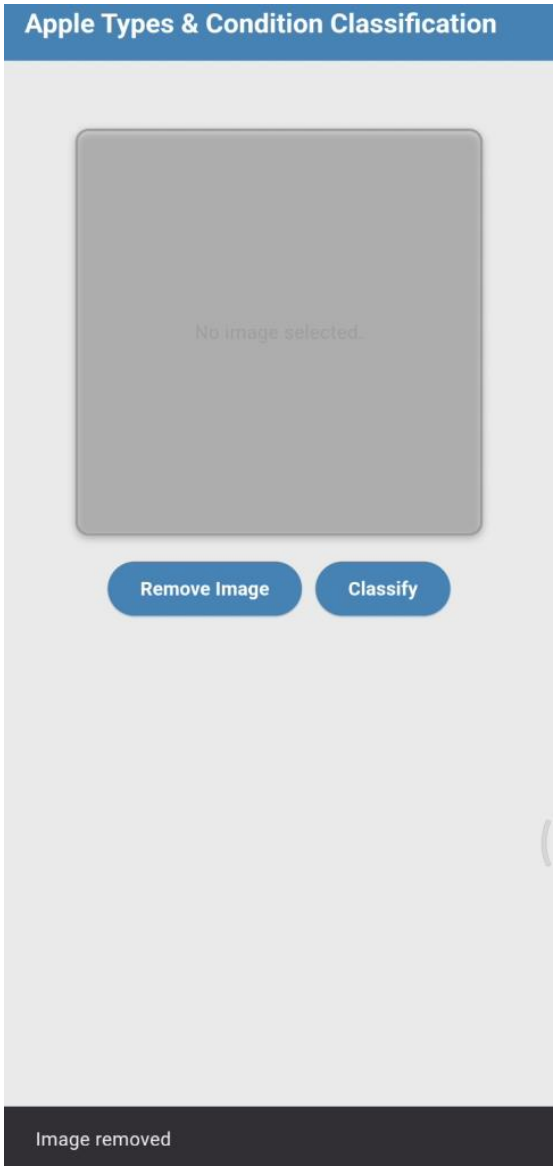


Figure 5.4.5 Image removed notification.

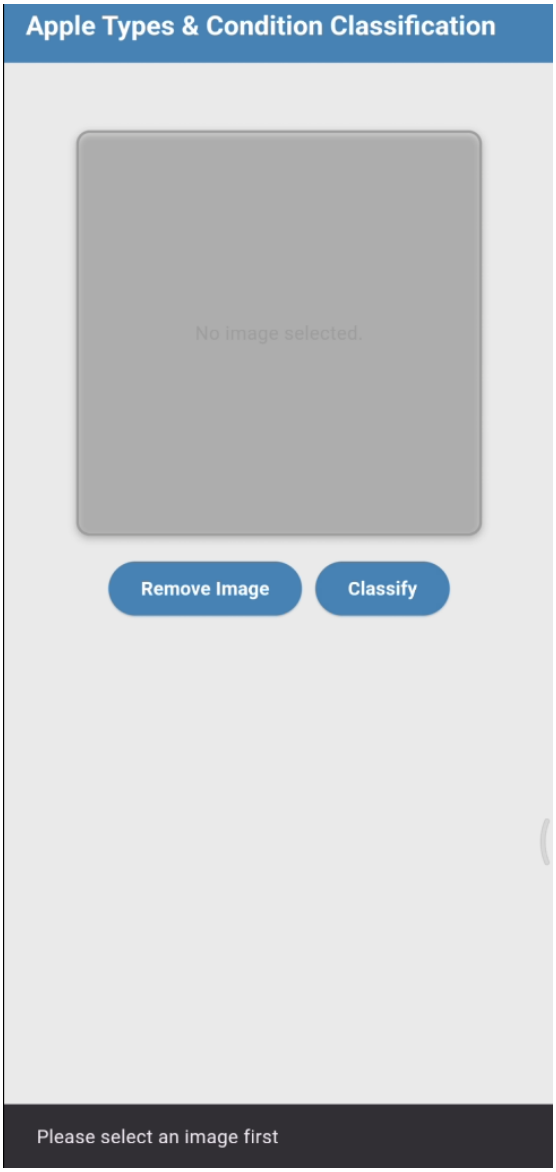


Figure 5.4.6 Select an image first notification.

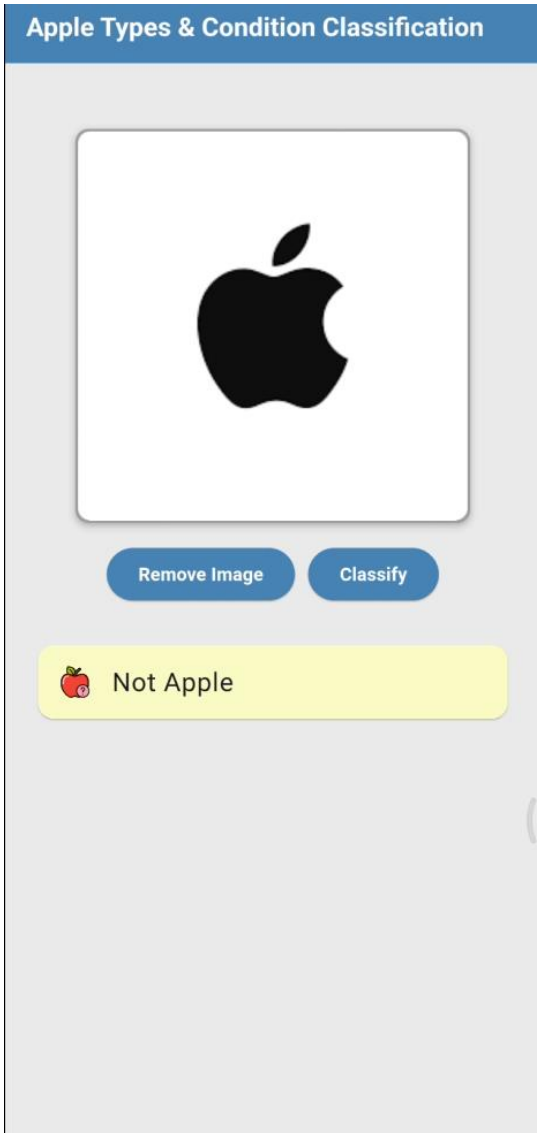


Figure 5.4.7 Not apple classification result.

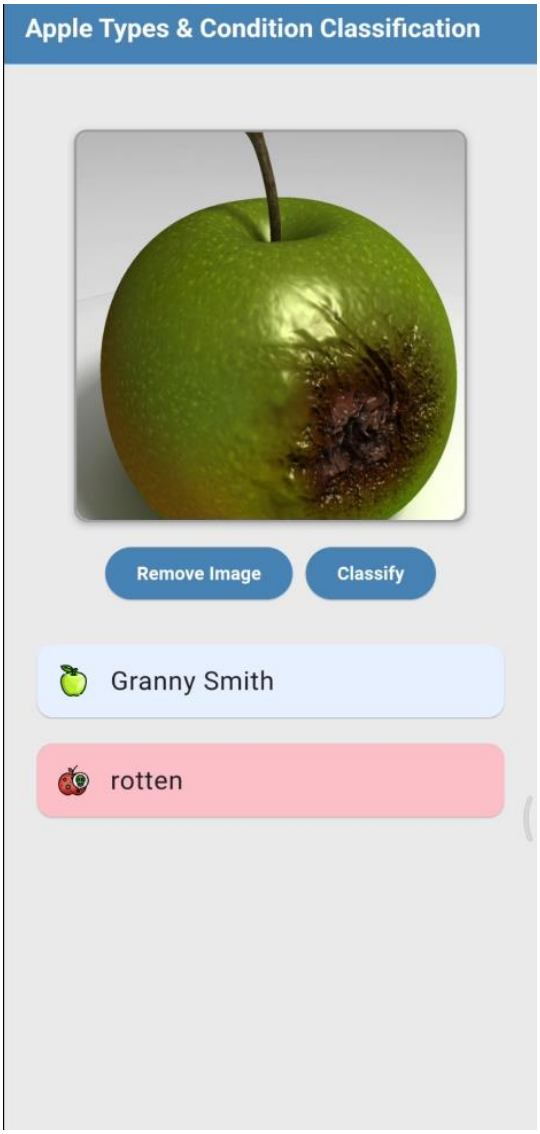


Figure 5.4.8 Rotten apple classification result.

5.5 Implementation Issues and Challenges

A major challenge encountered during the implementation of this project was the limited availability of the dataset. The dataset only contained 2181 images for different apple types (Cripps Red, Fuji, Granny Smith, Red Delicious, not apple) and 1823 images for apple conditions (fresh, rotten). This limited dataset size was inadequate for the models to effectively learn the necessary features and patterns. Since deep learning models are "data hungry" and require substantial amounts of data to learn and generalize well to unseen data, the limited size of the dataset increased the risk of overfitting. This would result in the model memorizing the

training data rather than genuinely learning from it. Although data augmentation techniques can help mitigate this issue, their effectiveness is often limited when dealing with such a small dataset.

In the model development phase, several implementation issues and challenges emerged. One of the main challenges was selecting an appropriate model that balanced accuracy and computational efficiency. Designing custom models also required careful consideration of model complexity and computational constraints. Additionally, determining optimal hyperparameters such as batch size, learning rate, regularization techniques, and the number of epochs was a significant challenge. It often required extensive experimentation, time and effort for fine-tuning.

Developing the mobile application also presented several challenges. One major issue was keeping up with the rapid updates in Flutter libraries, packages, and dependencies. Since the Flutter ecosystem evolves quickly, online tutorials and resources often become outdated, making it difficult to find relevant and up-to-date references. Moreover, while many online resources focus on implementing a single model, this project required loading two separate models to classify both apple types and conditions, which added to be the complexity. Another significant challenge involved deprecated dependencies and compatibility issues. For instance, the “tflite_flutter” and “tflite_flutter_helper” dependencies were initially considered because both could load multiple models simultaneously. However, both of these dependencies are deprecated and no longer supported. While “flutter_tflite” dependency worked for loading a single model, it did not support loading multiple models at the same time, which is critical to this project. Lastly, version conflicts between dependencies posed additional challenges. While it is generally advisable to use the latest versions of all dependencies in Flutter, this can lead to conflicts. For example, using the dependencies “image: ^3.0.1”, “image_picker: ^0.8.9’, and “flutter_tflite: ^1.0.1” worked well because none of these were the latest versions. However, updating these dependencies to their latest versions resulted in conflicts. For example, “image” from version 3.0.1 to 4.0.1 and “image_picker” from 0.8.9 to 1.1.2. As a result, considerable time was spent finding compatible dependencies on official documentation websites, which is time-consuming and complex.

5.6 Concluding Remark

Despite numerous issues and challenges encountered during the project, all of them were successfully resolved. While “flutter_tflite” dependency does not support loading multiple models simultaneously, this limitation was managed by sequentially loading one model at a time and unloading it before loading the second model. Finding compatible dependencies took considerable time, but online resources and official documentation were instrumental in overcoming these compatibility issues. Additionally, although many tutorial videos found online were outdated and lacked full support, they still offered valuable insights into the concepts and methods required for implementation. Overall, the difficulties faced during the project were effectively managed, thanks to the availability of online resources, comprehensive documentation, and the guidance and ideas provided by my supervisor, Dr Muhammad Syaiful Amri Bin Suhaimi.

Chapter 6
System Evaluation And Discussion

6.1 System Testing and Performance Metrics

System testing was a crucial phase in this project to ensure the application functioned as intended and met the desired requirements. The testing process involved validating the accuracy, reliability, and performance of both the deep learning models and the mobile application. To access the performance, several tools were employed, such as classification report, confusion matrix, and visualization of training and validation loss over epochs.

The classification report provided key metrics such as precision, recall, and F1-score for each class. This offers a comprehensive understanding of how well the models performed. On the other hand, the confusion matrix offered a visual representation of the classification results, highlighting the true positive, true negative, false positive, and false negative rates for each model, which helped identify specific areas of misclassification. The final model selection was based on a balance between accuracy and model size. A smaller model with high accuracy was preferred, as it ensured that the mobile application would run efficiently on various devices without compromising the quality of the classification results. The best model was chosen by using these performance metrics and visualizations. This approach is to ensure that the application would deliver accurate and reliable results in classifying apple types and conditions.

6.2 Testing Setup and Result

The system's effectiveness was evaluated by testing three different proposed models in this project. To evaluate the performance of the trained models on the test set, the code is written as shown in Figure 6.2.1. To summarize the performance of each model in classifying apple types and conditions, table 6.2.1 shows the summary.

Figure 6.2.1 Test loss and accuracy of apple types (left) and conditions model (right).

Table 6.2.1 Performance of proposed model.

Model	**Apple Types**		**Apple Conditions**	
	Test Loss	**Test Accuracy (%)**	**Test Loss**	**Test Accuracy (%)**
Self-defined CNN	0.0291	99.31	0.1953	92.88
MobileNetV2 (without weights)	1.6094	17.39	0.6932	48.49
Pre-trained MobileNetV2	0.0258	99.77	0.0828	96.44

To better understand the model's performance, code in Figure 6.2.2 visualizes the model's predictions. The code snippet randomly selects fifteen images from the test set and displays its actual and predicted labels. The title color of each image indicates the prediction's accuracy, green for correct predictions and red for incorrect ones. As the visualization

process for each model prediction on test set is consistent, therefore the best model (MobileNetV2 with pretrained weight) is chosen for demonstration. Figure 6.2.3 and Figure 6.2.4 provides the visualization of model predictions.

```
# Display 15 random pictures from the dataset with their labels
random_index = np.random.randint(0, len(test_df) - 1, 15)
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(25, 15),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(test_df.Filepath.iloc[random_index[i]]))
    if test_df.Label.iloc[random_index[i]] == pred_labels[random_index[i]]:
        color = "green"
    else:
        color = "red"
    ax.set_title(f"Actual: {test_df.Label.iloc[random_index[i]]}\nPredicted: {pred_labels[random_index[i]]}", color=color)
plt.show()
plt.tight_layout()
```

Figure 6.2.2 Visualization of predictions.

Figure 6.2.3 MobileNetV2 predictions (apple types).

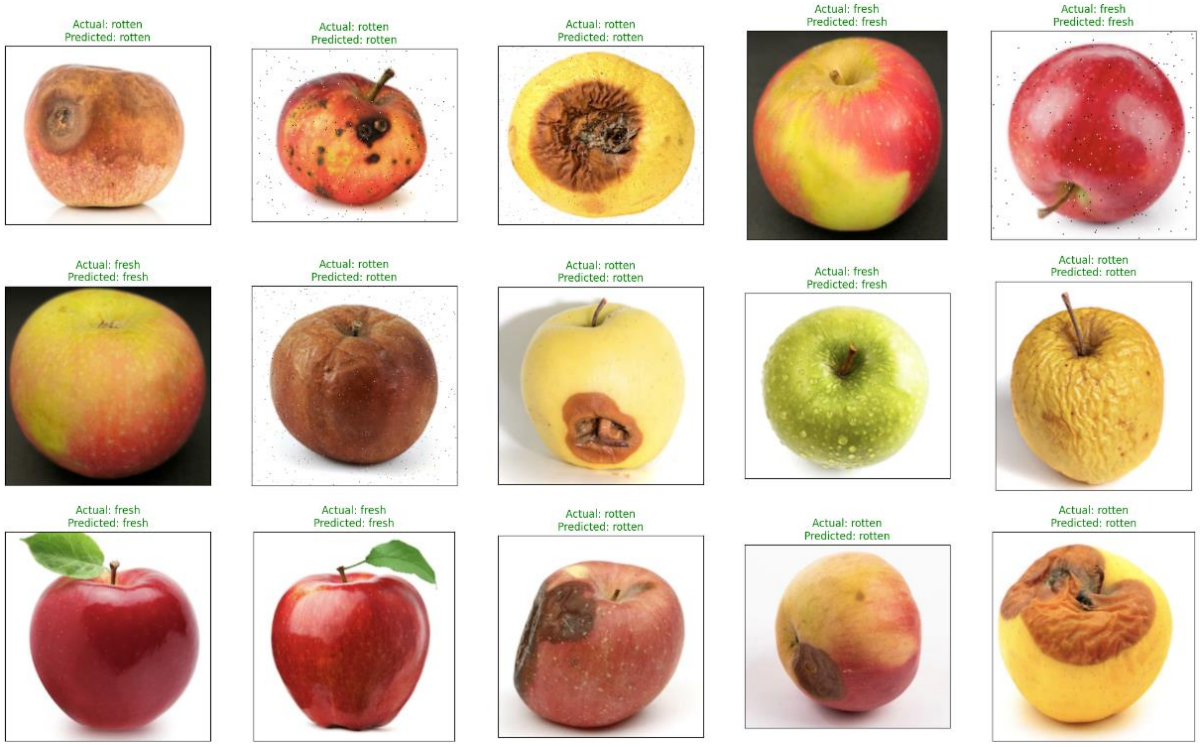
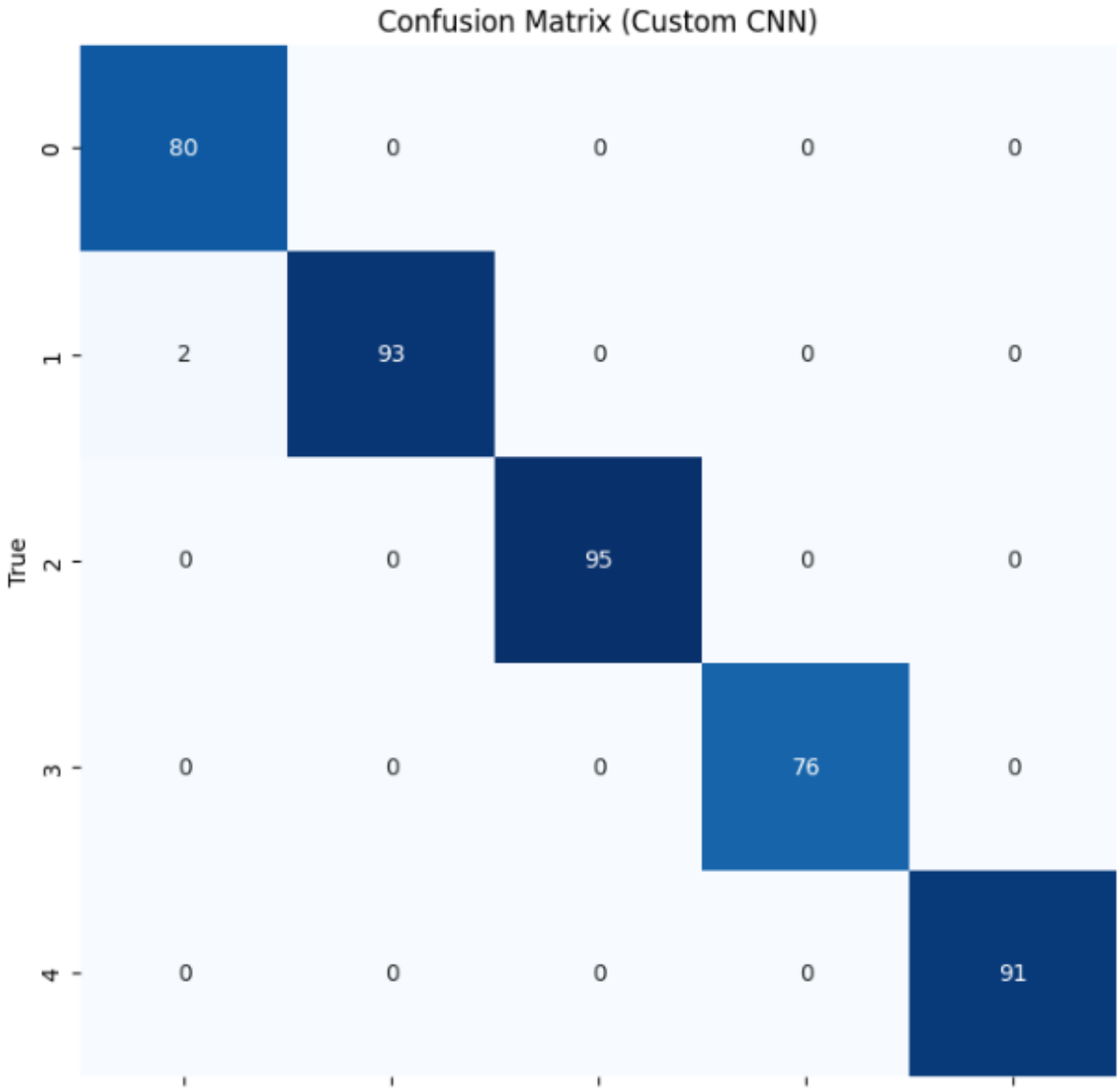


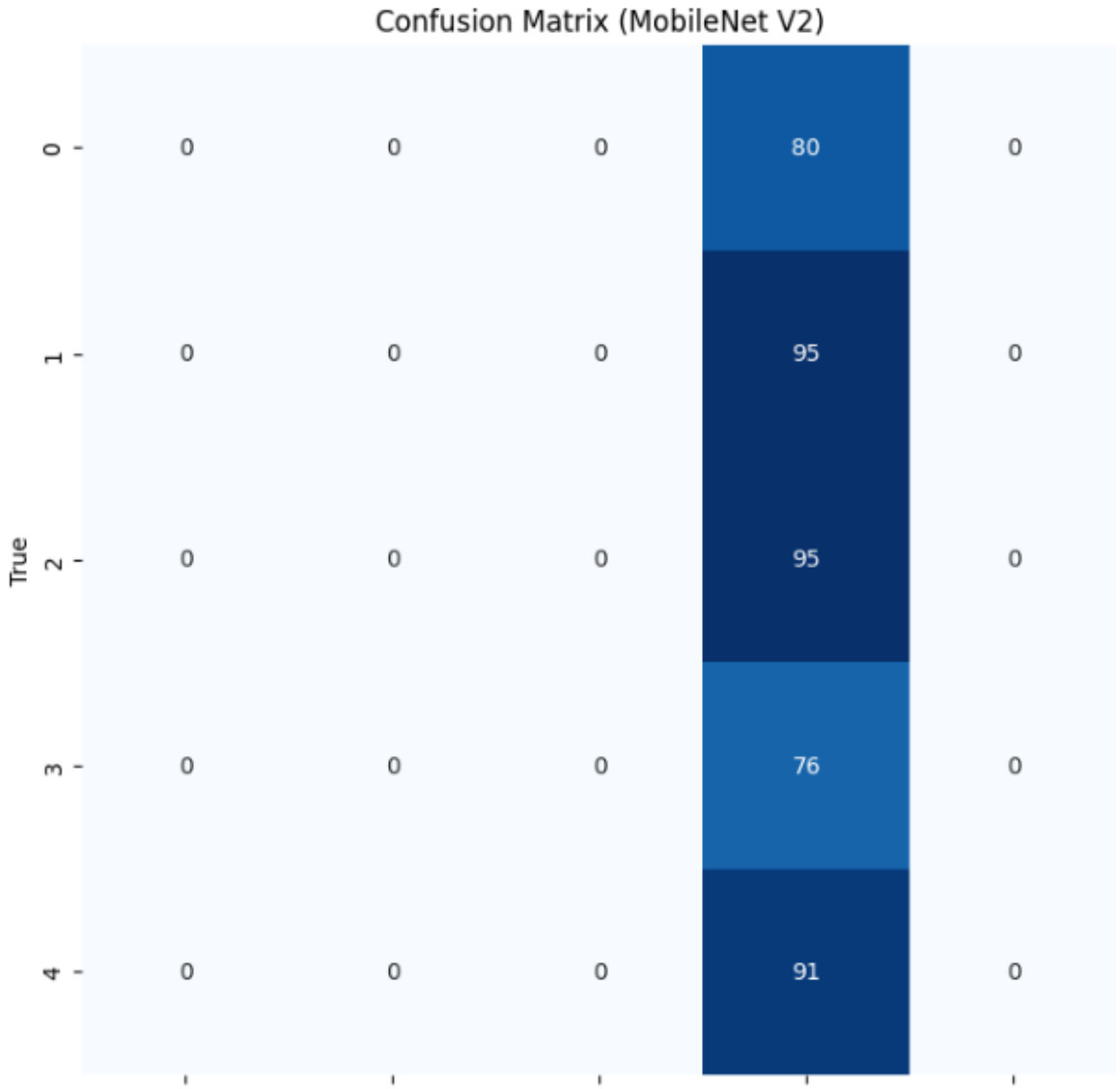
Figure 6.2.4 MobileNetV2 predictions (apple conditions).

A confusion matrix was generated for each model to visualize how well the model classified different classes. It highlights the number of true positives, true negatives, false positives, and false negatives to provide insight into the model's accuracy and error types. Alongside the confusion matrix, a detailed classification report was generated to evaluate the precision, recall, and F1-score of each model. This report offers a comprehensive overview of each model's performance across different classes. Screenshots of the confusion matrix and classification report of each model is provided in Figure 6.2.5.



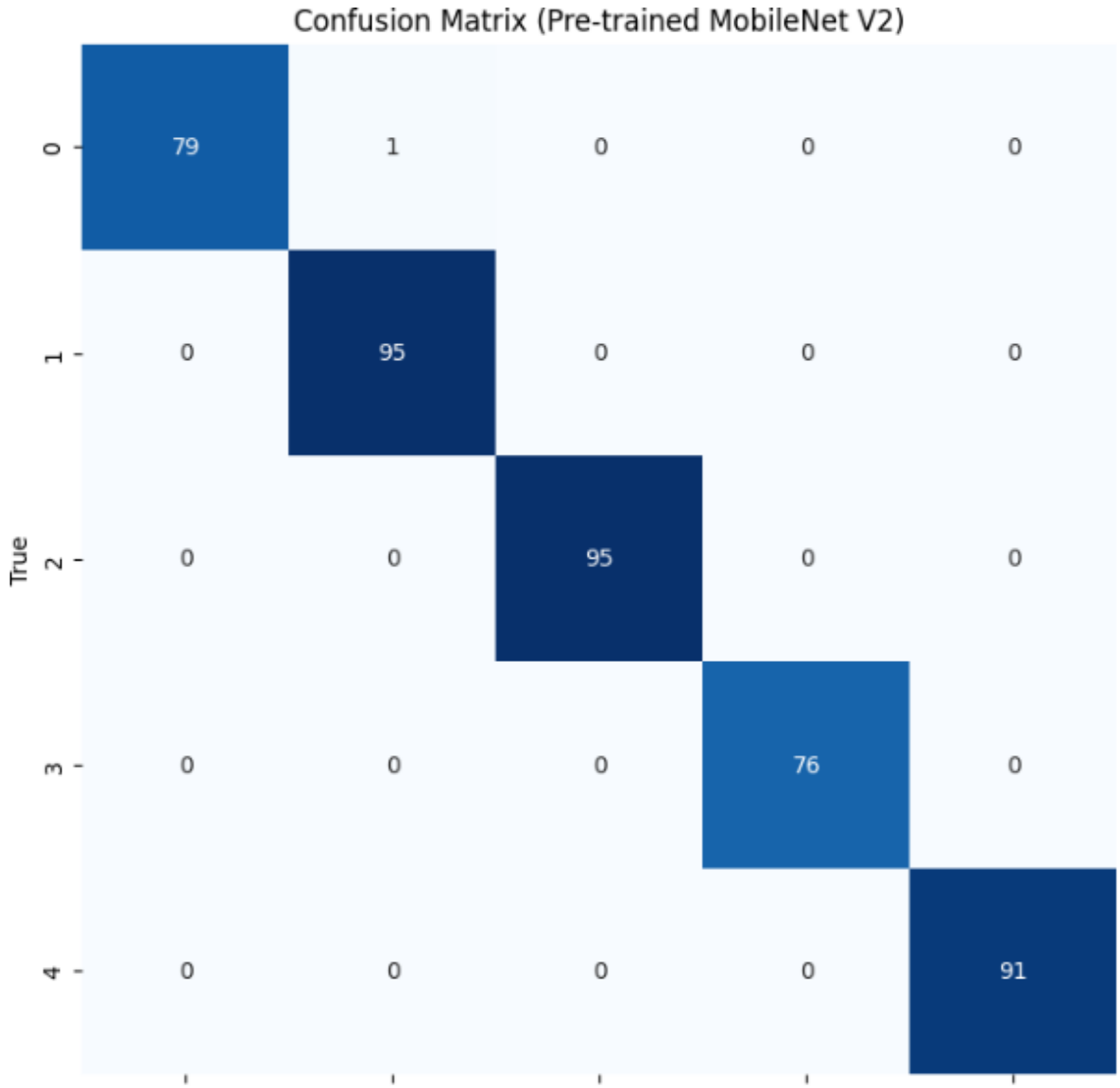
	precision	recall	f1-score	support
Cripps Red	0.98	1.00	0.99	80
Fuji	1.00	0.98	0.99	95
Granny Smith	1.00	1.00	1.00	95
Not Apple	1.00	1.00	1.00	76
Red Delicious	1.00	1.00	1.00	91
accuracy			1.00	437
macro avg	1.00	1.00	1.00	437
weighted avg	1.00	1.00	1.00	437

Figure 6.2.5 Confusion matrix and classification report of self-defined CNN (apple types).



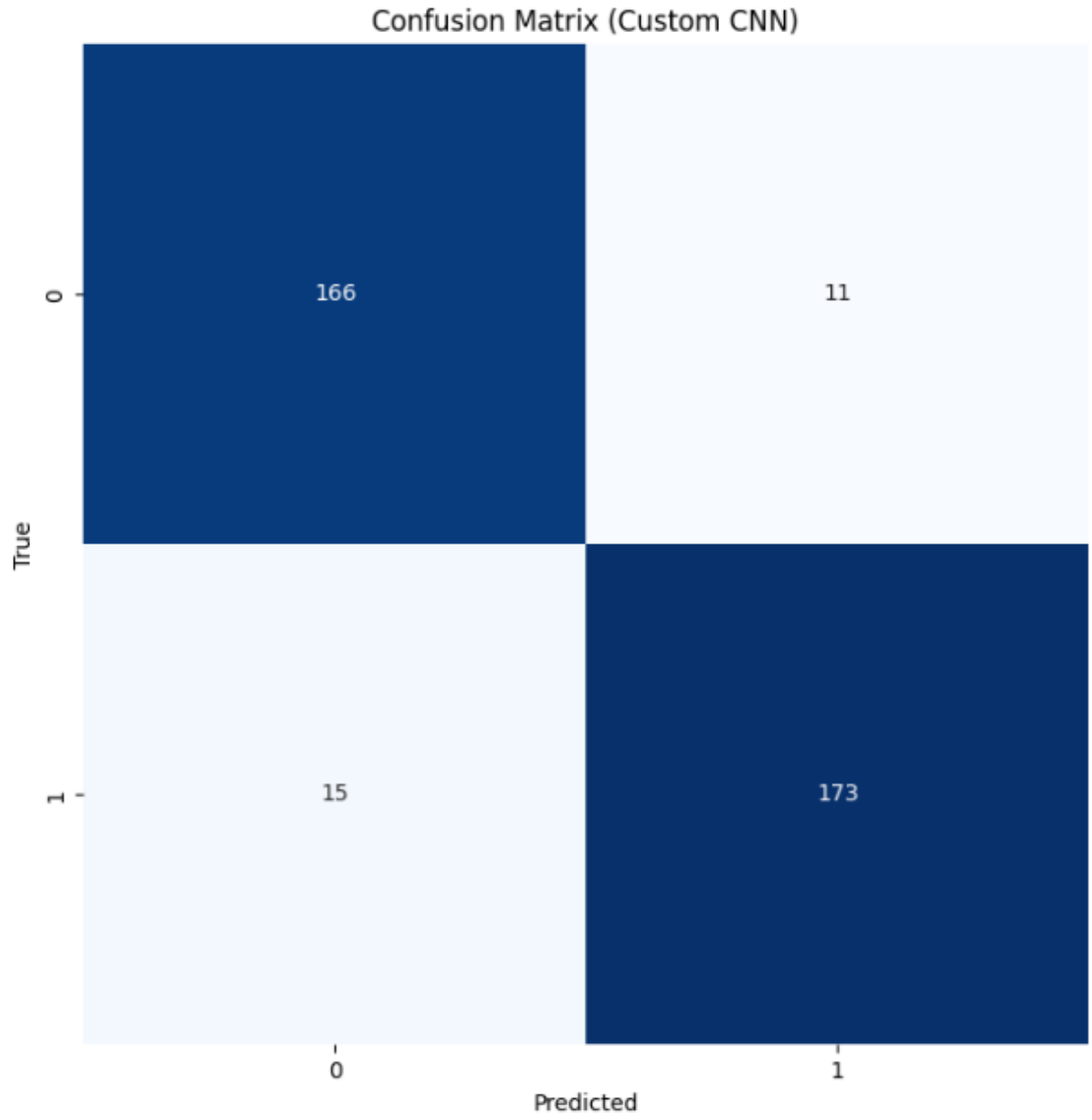
	precision	recall	f1-score	support
Cripps Red	0.00	0.00	0.00	80
Fuji	0.00	0.00	0.00	95
Granny Smith	0.00	0.00	0.00	95
Not Apple	0.17	1.00	0.30	76
Red Delicious	0.00	0.00	0.00	91
accuracy			0.17	437
macro avg	0.03	0.20	0.06	437
weighted avg	0.03	0.17	0.05	437

Figure 6.2.6 Confusion matrix and classification report of MobileNetV2 (apple types).



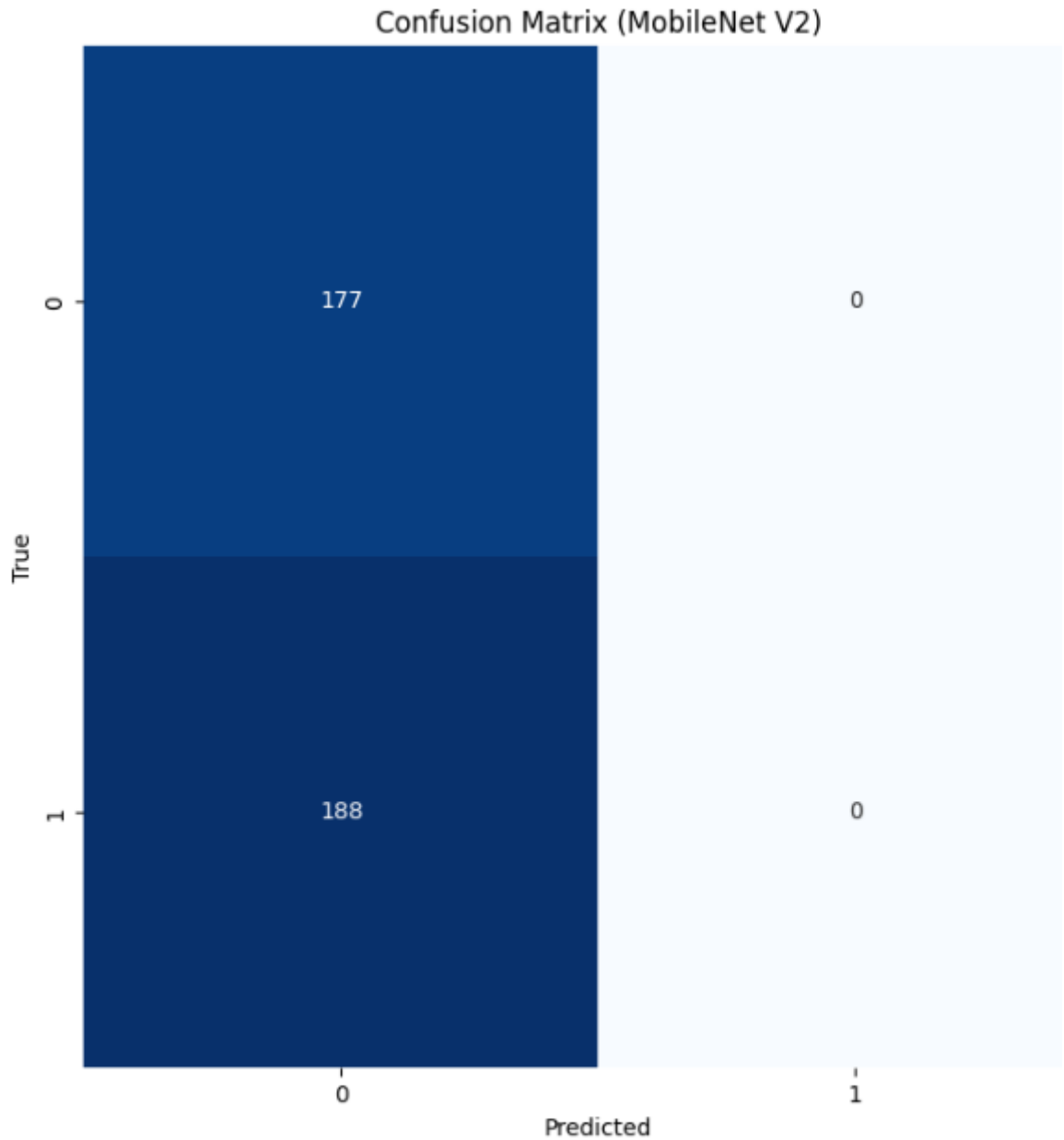
	precision	recall	f1-score	support
Cripps Red	1.00	0.99	0.99	80
Fuji	0.99	1.00	0.99	95
Granny Smith	1.00	1.00	1.00	95
Not Apple	1.00	1.00	1.00	76
Red Delicious	1.00	1.00	1.00	91
accuracy			1.00	437
macro avg	1.00	1.00	1.00	437
weighted avg	1.00	1.00	1.00	437

Figure 6.2.7 Confusion matrix and classification report of pretrained MobileNetV2 (apple types).



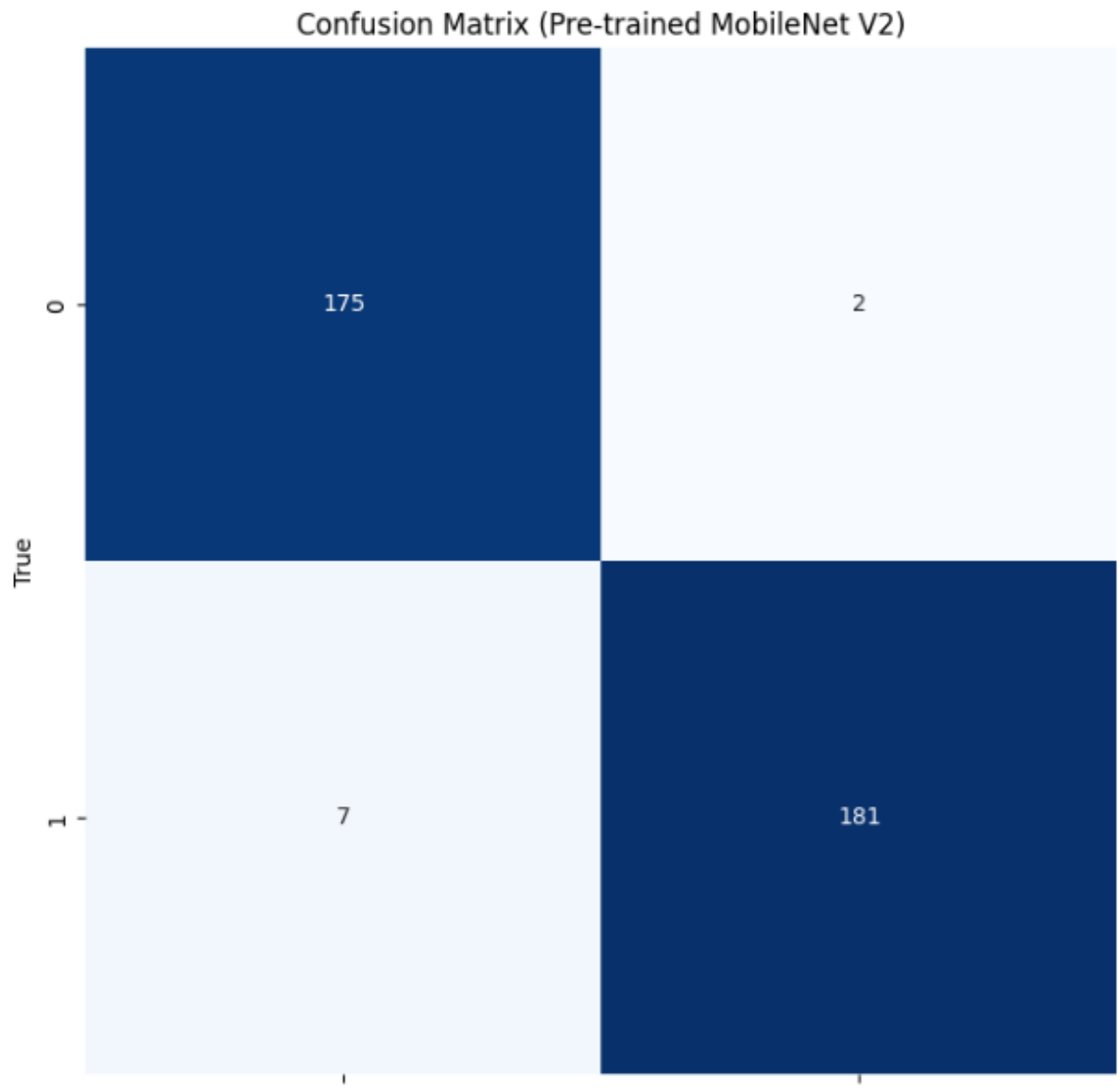
	precision	recall	f1-score	support
fresh	0.92	0.94	0.93	177
rotten	0.94	0.92	0.93	188
accuracy			0.93	365
macro avg	0.93	0.93	0.93	365
weighted avg	0.93	0.93	0.93	365

Figure 6.2.8 Confusion matrix and classification report of self-defined CNN (apple conditions).



	precision	recall	f1-score	support
fresh	0.48	1.00	0.65	177
rotten	0.00	0.00	0.00	188
accuracy			0.48	365
macro avg	0.24	0.50	0.33	365
weighted avg	0.24	0.48	0.32	365

Figure 6.2.9 Confusion matrix and classification report of MobileNetV2 (apple conditions).



	precision	recall	f1-score	support
fresh	0.96	0.99	0.97	177
rotten	0.99	0.96	0.98	188
accuracy			0.98	365
macro avg	0.98	0.98	0.98	365
weighted avg	0.98	0.98	0.98	365

Figure 6.2.10 Confusion matrix and classification report of pretrained MobileNetV2 (apple conditions).

Last but not least, the models were also tested in real-world scenarios using Jupyter Notebook and mobile application. The performance of the models was observed by uploading or capturing real-world images of apples and evaluating the classification results. Initially, all

three proposed models were tested in Jupyter Notebook to determine the best-performing model. The best model from this evaluation was then deployed to the mobile application. This section will only showcase system testing conducted in Jupyter Notebook, as system testing in the mobile app has already been discussed in the previous chapter (5.4 System Operation). Due to the repetitive nature of the results, this section will only present the real-world test results of the best model for both types and condition classification. To set up the real-world testing, the code in Figure 6.2.11 was used to load and preprocess the image. Figure 6.2.12 and 6.2.13 shows the real-world classification result.

```
# Load and preprocess the image
img_path = 'testing/t2.jpg'
img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size as needed
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
img_array /= 255.  # Normalize pixel values

# Predict the label
probs = model3.predict(img_array)
pred_label_idx = np.argmax(probs)
pred_label = labels[pred_label_idx]

# Display the image and predicted label
plt.imshow(img)
plt.title(f'Predicted Label: {pred_label}')
plt.axis('off')
plt.show()
```

Figure 6.2.11 Code for Real-World Testing in Jupyter Notebook.

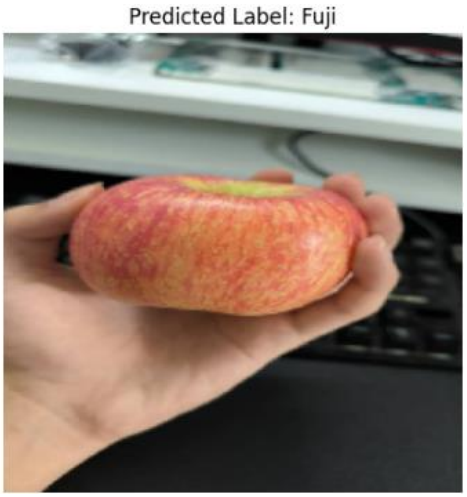


Figure 6.2.12 Real-world classification result in Jupyter Notebook (apple types).

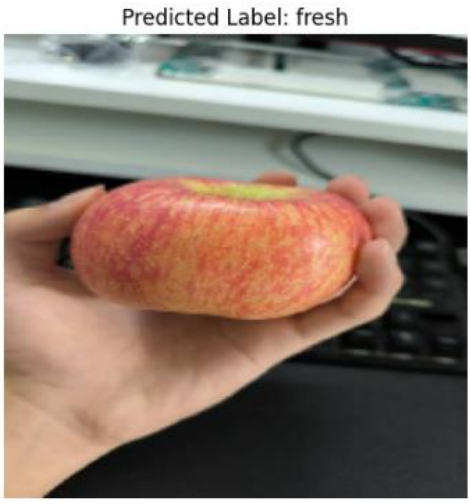


Figure 6.2.13 Real-world classification result in Jupyter Notebook (apple conditions).

6.3 Project Challenges

The unavailability of a comprehensive dataset, especially for apple types, necessitated manual data collection to address the issue. This involved physically purchasing and searching for various apple varieties, photographing each type under controlled conditions to ensure image consistency. However, this manual data collection process presented challenges, such as inaccurate categorization and misplacement of apple types in supermarkets, leading to increased time and effort during the purchasing phase. After acquiring the images, the subsequent steps of cropping and processing each image added to the challenges. Each image had to be carefully cropped to focus solely on the apple, removing unnecessary background elements or surrounding objects. Once the image processing was complete, each processed image needs to be accurately moved to its corresponding folder based on the apple variety. Improper cropping and labelling could significantly impact model performance during training. Overall, the task of physically collecting data and processing images individually required a substantial investment of time and effort. On the other hand, finding a dataset for apple conditions was also challenging. Manual collection was not practical as it required waiting a long time for apples to naturally rot. Additionally, there is seldom research or focus on apple conditions, which results in fewer datasets available online. Consequently, only a total of 1823 images were found for apple condition classification.

6.4 Objectives Evaluation

The primary objective was to develop a robust deep learning model capable of classifying apple types and conditions in real-world scenarios. Although the model cannot be considered entirely "robust" due to occasional misclassifications, it successfully classifies apple types and conditions most of the time. However, the model sometimes fails to classify certain images. For example, there are instances where there is no output when the “Classify” button is clicked, for reasons that remain unclear. Additionally, if the image background is too complex, the model tends to bias the classification towards the Red Delicious apple type. Despite these limitations, the, the results are generally accurate, which are making the model reliable for practical use cases. Therefore, this objective can be considered partially achieved, as the model demonstrates good performance in real-world scenarios despite not being flawless.

Another key objective was to optimize the deep learning model for deployment on mobile devices. This objective is considered fully achieved. The best-performing model, MobileNetV2

with pretrained weights is only 8.61MB in size, which is significantly smaller compared to the self-defined CNN model, which is 84.98MB. Despite its smaller size, the MobileNetV2 model achieves higher accuracy, making it both lightweight and efficient for mobile deployment. The model runs smoothly on mobile devices without causing significant resource strain, and therefore confirming the successful optimization for mobile deployment.

The third objective focused on enhancing the user experience by implementing a user-friendly interface for the application. This objective has been successfully met. The application's UI is straightforward and intuitive, allowing users to see the main components directly. It features a prominent grey box where users can select an image either from the gallery or by taking a picture. Additionally, there are two buttons for "Remove Image" and "Classify," making the interface simple and easy to use. The overall design is clean with adequately sized fonts that are easy to read in order to ensure a user-friendly experience for the consumers.

6.5 Concluding Remark

The objectives were met effectively with the deep learning model demonstrating the capability to classify apple types and conditions accurately in most real-world scenarios. The testing proven that the system functions as expected and provided reliable and adequate results. In summation, this project achieved its end goal of creating a practical tool that delivers accurate classifications, thereby providing valuable assistance to consumers in their daily lives.

Chapter 7
Conclusion and Recommendation

7.1 Conclusion

In conclusion, this project successfully developed a mobile application that enables consumers to assess apple types and conditions using deep learning models. Despite facing challenges such as limited dataset availability and dependency management, the objectives were largely achieved. The application demonstrated the ability to classify apple types and conditions accurately in real-world scenarios, providing valuable insights to users. The optimization of the model for mobile deployment ensured efficient performance with the best model being lightweight and running smoothly on the mobile devices. Additionally, the user-friendly interface enhanced the overall user experience, making the application accessible and easy to use. In summary, this project met its goals of creating a practical tool that enhances consumer decision-making and contributes positively to everyday life.

7.2 Recommendation

First and foremost, expanding the dataset is vital to enhance the robustness and accuracy of the deep learning model. This can be achieved through collecting more images under varied conditions, such as different backgrounds and lightning. At this point, the model can better generalize its classifications and reduce biases towards specific scenarios. Additionally, incorporating a wider variety of apple types into the dataset would further improve the model's ability to classify different apple varieties accurately. This not only increases the model's accuracy but also makes the application more useful for a broader range of consumers who encounter different apple types in their daily lives.

Another recommendation is to enhance the model's robustness by focusing on its ability to handle complex backgrounds and consistently provide accurate results. Future work could involve using more advanced deep learning techniques, such as transfer learning with more sophisticated models or applying ensemble methods. In addition, experimenting with different models, such as MobileNetV3, could also provide further improvements in term of performance and efficiency. These approaches could help the model to better manage the variability in real-world images and improve its overall performance.

Moreover, integrating real-time classification capabilities could significantly enhance user experience. While the current application supports image upload and photo capture for classification, adding real-time analysis would provide immediate feedback and make the application even more interactive. This would require further optimization of the model to ensure it can process real-time data efficiently without compromising performance.

Apart from that, expanding the application to support other platforms such as IOS and web-based versions, could increase its accessibility and usability across a broader range of devices. Hence, more users could benefit from the tool by making the application available on multiple platforms, thereby increasing its impact and reach. Cross-platform compatibility would make the application more versatile and accessible.

Finally, implementing a user feedback feature could be highly beneficial. Allowing users to provide feedback on the classification results can help identify areas where the model may need improvement. This feedback can be utilized to periodically retrain the model, thereby refining its accuracy and reliability over time. Engaging users in the refinement process not only improves the model's performance but also enhances user trust and satisfaction with the application.

REFERENCES

[1] A. G. Howard et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017. https://doi.org/10.48550/arXiv.1704.04861

[2] B. Biswas et al., “A robust multi-label fruit classification based on Deep Convolution Neural Network,” Computational Intelligence in Pattern Recognition, pp. 105–115, 2019. doi:10.1007/978-981-13-9042-5_10

[3] F. Ali et al., "Fruits Classification using Convolutional Neural Network," Global Research and Development Journal for Engineering, vol. 5, no. 8, pp. 1-7, 2020.

[4]IBM, “What is Deep Learning?,” www.ibm.com, 2023. https://www.ibm.com/topics/deep-learning

[5] P. Kanupuru and N. V. Reddy, “A deep learning approach to detect the spoiled fruits,” WSEAS TRANSACTIONS ON COMPUTER RESEARCH, vol. 10, pp. 74–87, 2022. doi:10.37394/232018.2022.10.10

[6] T. B. Kumar et al., “A Novel Model to Detect and Classify Fresh and Damaged Fruits to Reduce Food Waste Using a Deep Learning Technique”, Journal of Food Quality, vol. 2022, Article ID 4661108, 8 pages, 2022. https://doi.org/10.1155/2022/4661108

[7] U. Shruthi et al., “Apple Varieties Classification using Light Weight CNN Model,” 2022 4th International Conference on Circuits, Control, Communication and Computing (I4C), Bangalore, India, 2022, pp. 68-72, doi: 10.1109/I4C57141.2022.10057703.

[8] S. Chakraborty et al, “Implementation of Deep Learning Methods to Identify Rotten Fruits,” 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2021, pp. 1207-1212, doi: 10.1109/ICOEI51242.2021.9453004.

[9] S. S. Palakodati et al., “Fresh and rotten fruits classification using CNN and transfer learning,” Revue d’Intelligence Artificielle, vol. 34, no. 5, pp. 617–622, 2020. doi:10.18280/ria.340512

[10] Y. Li et al., “Apple quality identification and classification by image processing based on Convolutional Neural Networks,” Scientific Reports, vol. 11, no. 1, 2021. doi:10.1038/s41598-021-96103-2

[11] Y. Wang et al., “A survey on deploying Mobile Deep Learning Applications: A Systemic and Technical Perspective,” Digital Communications and Networks, vol. 8, no. 1, pp. 1–17, 2022. doi:10.1016/j.dcan.2021.06.001

APPENDIX

```
# tensorflow libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Activation, Add, BatchNormalization, Conv2D, Flatten, Dense, DepthwiseConv2D, Dropout, GlobalAveragePooling2D, Input,
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import Callback, EarlyStopping,ModelCheckpoint, ReduceLROnPlateau

# system libraries
import os
import random
from pathlib import Path

# visualization libraries
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import cv2
import seaborn as sns

import pandas as pd
import numpy as np

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, make_scorer
from sklearn.model_selection import train_test_split
```

```
# Define constants
DATASET_PATH = "C:\\Users\\User\\Downloads\\FYP-test\\ds\\"
BATCH_SIZE = 32
EPOCHS = 20
TARGET_SIZE = (224, 224)
```

```
def walk_through_dir(dir_path):
  for dirpath, dirnames, filenames in os.walk(dir_path):
    print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
```

```
def load_dataset(file):
    data = []
    class_labels = []
    for label, class_folder in enumerate(os.listdir(file)):
        class_labels.append(class_folder)
        class_path = os.path.join(file, class_folder)
        if os.path.isdir(class_path):
            for image_file in os.listdir(class_path):
                image_path = os.path.join(class_path, image_file)
                # Append (file_path, label) tuple to data list
                data.append((image_path, label))
    return data, class_labels
```

```
# Walk through each directory for apple types
walk_through_dir(DATASET_PATH);
```

```
image, labels = load_dataset(DATASET_PATH)
print(image)
print(labels)
```

```
image_dir = Path(DATASET_PATH)

# Get filepaths and labels
filepaths = list(image_dir.glob(r'**/*.jpg'))

type_labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], filepaths))

filepaths = pd.Series(filepaths, name='Filepath').astype(str)

type_labels = pd.Series(type_labels, name='Label')

# Concatenate filepaths and labels
image_df = pd.concat([filepaths, type_labels], axis=1)

print(type_labels)
```

```
label_counts = image_df['Label'].value_counts()[:20]

plt.figure(figsize=(20, 6))
sns.barplot(x=label_counts.index, y=label_counts.values, alpha=0.8, palette='Blues')
plt.title('Distribution of labels in Apple Types Dataset', fontsize=16)
plt.xlabel('Label', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=45)
plt.show()
```

```
random_index = np.random.randint(0, len(image_df), 16)
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(10, 10),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(image_df.Filepath[random_index[i]]))
    ax.set_title(image_df.Label[random_index[i]])
plt.tight_layout()
plt.show()
```

```
# Separate in train and test data
train_df, test_df = train_test_split(image_df, test_size=0.2, shuffle=True, random_state=42)
```

```
# Set up data generators for training, validation, and testing
train_datagen = ImageDataGenerator(
    rescale=1./255,             # Rescale pixel values to [0, 1]
    validation_split=0.2,       # Split 20% of the data for validation
    rotation_range=20,          # Randomly rotate images by up to 20 degrees
    width_shift_range=0.1,      # Randomly shift images horizontally by up to 10%
    height_shift_range=0.1,     # Randomly shift images vertically by up to 10%
    shear_range=0.1,            # Apply shear transformation with intensity up to 10%
    zoom_range=0.1,             # Randomly zoom into images by up to 10%
    horizontal_flip=True,       # Randomly flip images horizontally
    fill_mode='nearest'         # Strategy for filling in newly created pixels after rotation or shifting
)

# Create a generator for training data
train_generator = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',  # Column containing the file paths
    y_col='Label',  # Column containing the labels
    target_size=TARGET_SIZE,
    color_mode='rgb',
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=True,  # Shuffle the data
    seed = 42,
    subset='training'
)

# Create a generator for validation data
validation_generator = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',  # Column containing the file paths
    y_col='Label',  # Column containing the labels
    target_size=TARGET_SIZE,
    color_mode='rgb',
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle=True,  # Shuffle the data
    seed = 42,
    subset='validation'
)

# Create a generator for testing data (if needed)
test_generator = train_datagen.flow_from_dataframe(
    dataframe=test_df,
    x_col='Filepath',  # Column containing the file paths
    y_col='Label',  # Column containing the labels
    target_size=TARGET_SIZE,
    color_mode='rgb',
    batch_size=BATCH_SIZE,
    class_mode='categorical',
    shuffle= False
)

# Display the class indices
print("Class Indices:", train_generator.class_indices)

# Display the number of samples in each split
print("Number of training samples:", train_generator.samples)
print("Number of validation samples:", validation_generator.samples)
print("Number of testing samples:", test_generator.samples)
```

```
# Fetch a batch of images and labels from the data generator
x_batch, y_batch = next(train_generator)

# Visualize the first few images and their labels
plt.figure(figsize=(10, 10))
for i in range(min(16, len(x_batch))):  # Visualize up to 16 images
    plt.subplot(4, 4, i + 1)
    plt.imshow(x_batch[i])
    plt.title('Label: ' + str(y_batch[i]))
    plt.axis('off')
plt.show()
```

```
print(y_batch)
```

```
# Define the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),  # Add dropout for regularization
    Dense(128, activation='relu'),
    Dropout(0.5),  # Add another dropout layer
    Dense(len(train_generator.class_indices), activation='softmax')  # Output layer with softmax activation for multi-class classification
])

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Display the model summary
model.summary()
```

```
base_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights=None)

# Freeze the base model
base_model.trainable = False

model2 = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    # tf.keras.layers.Dense(768, activation='relu'),
    # tf.keras.layers.Dense(768, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')  # Adjust the number of units to match the number of classes
])

model2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

model2.summary()
```

```
# Define the MobileNetV2 model with pre-trained weights from ImageNet
pretrained_mobilenetv2 = tf.keras.applications.MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights='imagenet')

# Freeze the weights of the pre-trained layers
pretrained_mobilenetv2.trainable = False

# Add a custom classification head on top of the pre-trained model
model3 = tf.keras.models.Sequential([
    pretrained_mobilenetv2,
    tf.keras.layers.GlobalAveragePooling2D(),
    # tf.keras.layers.Dense(768, activation='relu'),
    # tf.keras.layers.Dense(768, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')  # Adjusted number of units
])

# Compile the model
model3.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Display the model summary
model3.summary()
```

```
# Define callbacks
callbacks = [
    EarlyStopping(patience=5, monitor='val_loss', verbose=1),
    ModelCheckpoint('custom_CNN_type1.h5', save_best_only=True, monitor='val_loss', mode='min'),
    # ReduceLROnPlateau(factor=0.1, patience=2, min_lr=0.00001, verbose=1),
]

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    callbacks=callbacks
)
```

```
# Define callbacks
callbacks = [
    EarlyStopping(patience=5, monitor='val_loss', verbose=1),
    ModelCheckpoint('MobileNetv2_type2.h5', save_best_only=True, monitor='val_loss', mode='min')
    # ReduceLROnPlateau(factor=0.1, patience=2, min_lr=0.00001, verbose=1),
]

# Train the model
history = model2.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    callbacks=callbacks
)
```

```
# Define callbacks
callbacks = [
    EarlyStopping(patience=5, monitor='val_loss', verbose=1),
    ModelCheckpoint('MobileNetv2_type3.h5', save_best_only=True, monitor='val_loss', mode='min')
    # ReduceLROnPlateau(factor=0.1, patience=2, min_lr=0.00001, verbose=1),
]

# Train the model
history = model3.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    callbacks=callbacks
)
```

```
# Evaluate the trained model on the test set (run this when already done training)
model1 = load_model('custom_CNN_type1.h5')
test_loss, test_accuracy = model1.evaluate(test_generator)



# # # Evaluate the model on the test set
# # test_loss, test_accuracy = model2.evaluate(test_generator)

# Print the test loss and accuracy
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
##### import matplotlib.pyplot as plt

# Extracting training and validation accuracy and loss
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']


loss = history.history['loss']
val_loss = history.history['val_loss']


# Calculating the number of epochs
epochs = range(1, len(accuracy) + 1)

# Plotting accuracy
plt.plot(epochs, accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plotting loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
# Evaluate the trained model on the test set (run this when already done training)
model2 = load_model('MobileNetv2_type2.h5')
test_loss, test_accuracy = model2.evaluate(test_generator)

# # Evaluate the model on the test set
# test_loss, test_accuracy = model2_1.evaluate(test_generator)

# Print the test loss and accuracy
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
# Extracting training and validation accuracy and loss (model2_1)
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']


loss = history.history['loss']
val_loss = history.history['val_loss']


# Calculating the number of epochs
epochs = range(1, len(accuracy) + 1)

# Plotting accuracy
plt.plot(epochs, accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plotting Loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
# Evaluate the trained model on the test set (run this when already done training)
model3 = load_model('MobileNetv2_type3.h5')
test_loss, test_accuracy = model3.evaluate(test_generator)

# # Evaluate the model on the test set
# test_loss, test_accuracy = model2_2.evaluate(test_generator)

# Print the test loss and accuracy
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
# Extracting training and validation accuracy and loss (model_2)
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']


loss = history.history['loss']
val_loss = history.history['val_loss']


# Calculating the number of epochs
epochs = range(1, len(accuracy) + 1)

# Plotting accuracy
plt.plot(epochs, accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plotting loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
# Predict the label of the test_images
probs = model1.predict(test_generator)
pred = np.argmax(probs,axis=1)

# Map the label
labels = (train_generator.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred_labels = [labels[k] for k in pred]

# # Display the result
# print(pred[:5])
# print(f'The first 5 predictions: {pred_labels[:5]}')
```

```
# Display 15 random pictures from the dataset with their labels
random_index = np.random.randint(0, len(test_df) - 1, 15)
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(25, 15),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(test_df.Filepath.iloc[random_index[i]]))
    if test_df.Label.iloc[random_index[i]] == pred_labels[random_index[i]]:
        color = "green"
    else:
        color = "red"
    ax.set_title(f"Actual: {test_df.Label.iloc[random_index[i]]}\nPredicted: {pred_labels[random_index[i]]}", color=color)
plt.show()
plt.tight_layout()
```

```
# Get true labels from the test generator
actual = test_generator.classes

# Plot confusion matrix
cm = confusion_matrix(actual, pred)
plt.figure(figsize=(8, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix (Custom CNN)')
plt.show()

# Print classification report
class_labels = list(test_generator.class_indices.keys())
print(classification_report(actual, pred, target_names=class_labels))
```

```
# Predict the label of the test_images
probs2 = model2.predict(test_generator)
pred2 = np.argmax(probs2,axis=1)

# Map the label
labels2 = (train_generator.class_indices)
labels2 = dict((v,k) for k,v in labels2.items())
pred_labels2 = [labels2[k] for k in pred2]

# # Display the result
# print(pred[:5])
# print(f'The first 5 predictions: {pred_labels[:5]}')
```

```
# Display 15 random pictures from the dataset with their labels
random_index = np.random.randint(0, len(test_df) - 1, 15)
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(25, 15),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(test_df.Filepath.iloc[random_index[i]]))
    if test_df.Label.iloc[random_index[i]] == pred_labels2[random_index[i]]:
        color = "green"
    else:
        color = "red"
    ax.set_title(f"Actual: {test_df.Label.iloc[random_index[i]]}\nPredicted: {pred_labels2[random_index[i]]}", color=color)
plt.show()
plt.tight_layout()
```

```
# Get true labels from the test generator
actual = test_generator.classes

# Plot confusion matrix
cm = confusion_matrix(actual, pred2)
plt.figure(figsize=(8, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix (MobileNet V2)')
plt.show()

# Print classification report
class_labels = list(test_generator.class_indices.keys())
print(classification_report(actual, pred2, target_names=class_labels))
```

```
# Predict the label of the test_images
probs3 = model3.predict(test_generator)
pred3 = np.argmax(probs3,axis=1)

# Map the label
labels3 = (train_generator.class_indices)
labels3 = dict((v,k) for k,v in labels3.items())
pred_labels3 = [labels3[k] for k in pred3]
```

```
# Display 15 random pictures from the dataset with their labels
random_index = np.random.randint(0, len(test_df) - 1, 15)
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(25, 15),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(test_df.Filepath.iloc[random_index[i]]))
    if test_df.Label.iloc[random_index[i]] == pred_labels3[random_index[i]]:
        color = "green"
    else:
        color = "red"
    ax.set_title(f"Actual: {test_df.Label.iloc[random_index[i]]}\nPredicted: {pred_labels3[random_index[i]]}", color=color)
plt.show()
plt.tight_layout()
```

```
# Get true labels from the test generator
actual = test_generator.classes

# Plot confusion matrix
cm = confusion_matrix(actual, pred3)
plt.figure(figsize=(8, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix (Pre-trained MobileNet V2)')
plt.show()

# Print classification report
class_labels = list(test_generator.class_indices.keys())
print(classification_report(actual, pred3, target_names=class_labels))
```

```
# Load and preprocess the image
img_path = 'testing/reddelicioustest4.jpg'
img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size as needed
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
img_array /= 255.  # Normalize pixel values

# Predict the label
probs = model1.predict(img_array)
pred_label_idx = np.argmax(probs)
pred_label = labels[pred_label_idx]

# Display the image and predicted label
plt.imshow(img)
plt.title(f'Predicted Label: {pred_label}')
plt.axis('off')
plt.show()
```

```
# Load and preprocess the image
img_path = 'testing/t3.jpg'
img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size as needed
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
img_array /= 255.  # Normalize pixel values

# Predict the label
probs = model2.predict(img_array)
pred_label_idx = np.argmax(probs)
pred_label = labels[pred_label_idx]

# Display the image and predicted label
plt.imshow(img)
plt.title(f'Predicted Label: {pred_label}')
plt.axis('off')
plt.show()
```

```
# Load and preprocess the image
img_path = 'testing/t2.jpg'
img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size as needed
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
img_array /= 255.  # Normalize pixel values

# Predict the label
probs = model3.predict(img_array)
pred_label_idx = np.argmax(probs)
pred_label = labels[pred_label_idx]

# Display the image and predicted label
plt.imshow(img)
plt.title(f'Predicted Label: {pred_label}')
plt.axis('off')
plt.show()
```

```
# tensorflow libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Activation, Add, BatchNormalization, Conv2D, Flatten, Dense, DepthwiseConv2D, Dropout, GlobalAveragePooling2D, Input,
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import Callback, EarlyStopping,ModelCheckpoint, ReduceLROnPlateau

# system libraries
import os
import random
from pathlib import Path

# visualization libraries
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import cv2
import seaborn as sns

import pandas as pd
import numpy as np

from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, make_scorer
from sklearn.model_selection import train_test_split
```

```
# Define constants
DATASET_PATH = "C:\\Users\\User\\Downloads\\FYP-test\\condition_ds\\"
BATCH_SIZE = 32
EPOCHS = 20
TARGET_SIZE = (224, 224)
```

```
def walk_through_dir(dir_path):
  for dirpath, dirnames, filenames in os.walk(dir_path):
    print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")
```

```
def load_dataset(file):
    data = []
    class_labels = []
    for label, class_folder in enumerate(os.listdir(file)):
        class_labels.append(class_folder)
        class_path = os.path.join(file, class_folder)
        if os.path.isdir(class_path):
            for image_file in os.listdir(class_path):
                image_path = os.path.join(class_path, image_file)
                # Append (file_path, label) tuple to data list
                data.append((image_path, label))
    return data, class_labels
```

```
# Walk through each directory for apple conditions
walk_through_dir(DATASET_PATH)
```

```
image, labels = load_dataset(DATASET_PATH)
print(image)
print(labels)
```

```
image_dir = Path(DATASET_PATH)

# Get filepaths and labels
filepaths = list(image_dir.glob(r'**/*.jpg')) + list(image_dir.glob(r'**/*.png'))

condition_labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], filepaths))

filepaths = pd.Series(filepaths, name='Filepath').astype(str)

condition_labels = pd.Series(condition_labels, name='Label')

# Concatenate filepaths and labels
image_df = pd.concat([filepaths, condition_labels], axis=1)

print(condition_labels)
```

```
label_counts = image_df['Label'].value_counts()[:20]

plt.figure(figsize=(20, 6))
sns.barplot(x=label_counts.index, y=label_counts.values, alpha=0.8, palette='Blues')
plt.title('Distribution of labels in Apple Condition Dataset ', fontsize=16)
plt.xlabel('Label', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=45)
plt.show()
```

```
random_index = np.random.randint(0, len(image_df), 16)
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(10, 10),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(image_df.Filepath[random_index[i]]))
    ax.set_title(image_df.Label[random_index[i]])
plt.tight_layout()
plt.show()
```

```
# Separate in train and test data
train_df, test_df = train_test_split(image_df, test_size=0.2, shuffle=True, random_state=42)
```

```
# Set up data generators for training, validation, and testing
train_datagen = ImageDataGenerator(
    rescale=1./255,              # Rescale pixel values to [0, 1]
    validation_split=0.2,        # Split 20% of the data for validation
    rotation_range=15,           # Randomly rotate images by up to 20 degrees
    width_shift_range=0.1,       # Randomly shift images horizontally by up to 10%
    height_shift_range=0.1,      # Randomly shift images vertically by up to 10%
    shear_range=0.1,             # Apply shear transformation with intensity up to 10%
    zoom_range=0.1,              # Randomly zoom into images by up to 10%
    horizontal_flip=True,        # Randomly flip images horizontally
    fill_mode='nearest'          # Strategy for filling in newly created pixels after rotation or shifting
)


# Create a generator for training data
train_generator = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',  # Column containing the file paths
    y_col='Label',     # Column containing the labels
    target_size=TARGET_SIZE,
    color_mode='rgb',
    batch_size=BATCH_SIZE,
    class_mode='binary',  # For binary classification
    shuffle=True,  # Shuffle the data
    seed=42,
    subset='training'
)

# Create a generator for validation data
validation_generator = train_datagen.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepath',  # Column containing the file paths
    y_col='Label',     # Column containing the labels
    target_size=TARGET_SIZE,
    color_mode='rgb',
    batch_size=BATCH_SIZE,
    class_mode='binary',  # For binary classification
    shuffle=True,  # Shuffle the data
    seed=42,
    subset='validation'
)

# Create a generator for testing data (if needed)
test_generator = train_datagen.flow_from_dataframe(
    dataframe=test_df,
    x_col='Filepath',  # Column containing the file paths
    y_col='Label',     # Column containing the labels
    target_size=TARGET_SIZE,
    color_mode='rgb',
    batch_size=BATCH_SIZE,
    class_mode='binary',  # For binary classification
    shuffle=False  # No need to shuffle for testing
)

# Display the class indices
print("Class Indices:", train_generator.class_indices)

# Display the number of samples in each split
print("Number of training samples:", train_generator.samples)
print("Number of validation samples:", validation_generator.samples)
print("Number of testing samples:", test_generator.samples)
```

```
# Fetch a batch of images and labels from the data generator
x_batch, y_batch = next(train_generator)

# Visualize the first few images and their labels
plt.figure(figsize=(10, 10))
for i in range(min(16, len(x_batch))):  # Visualize up to 16 images
    plt.subplot(4, 4, i + 1)
    plt.imshow(x_batch[i])
    plt.title('Label: ' + str(y_batch[i]))
    plt.axis('off')
plt.show()
```

```
print(y_batch)
```

```python
# Define the model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(2, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Print model summary
model.summary()
```

```python
base_model = MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights=None)

# Freeze the base model
base_model.trainable = False

model2_1 = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    # tf.keras.layers.Dense(768, activation='relu'),
    # tf.keras.layers.Dense(768, activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')  # Adjust the number of units to match the number of classes
])

model2_1.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

model2_1.summary()
```

```python
# Define the MobileNetV2 model with pre-trained weights from ImageNet
pretrained_mobilenetv2 = tf.keras.applications.MobileNetV2(input_shape=(224, 224, 3), include_top=False, weights='imagenet')

# Freeze the weights of the pre-trained layers
pretrained_mobilenetv2.trainable = False

# Add a custom classification head on top of the pre-trained model
model2_2 = tf.keras.models.Sequential([
    pretrained_mobilenetv2,
    tf.keras.layers.GlobalAveragePooling2D(),
    # tf.keras.layers.Dense(768, activation='relu'),
    # tf.keras.layers.Dense(768, activation='relu'),
    tf.keras.layers.Dense(2, activation='softmax')  # Adjusted number of units
])

# Compile the model
model2_2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Display the model summary
model2_2.summary()
```

```python
callbacks = [
    EarlyStopping(patience=5, monitor='val_loss', verbose=1),
    ModelCheckpoint('custom_CNN_condition.h5', save_best_only=True, monitor='val_loss', mode='min'),
    ReduceLROnPlateau(factor=0.1, patience=2, min_lr=0.00001, verbose=1),
]

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch= train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    callbacks=callbacks
)
```

```python
callbacks = [
    EarlyStopping(patience=5, monitor='val_loss', verbose=1),
    ModelCheckpoint('MobileNetV2_condition2_1.h5', save_best_only=True, monitor='val_loss', mode='min')
    # ReduceLROnPlateau(factor=0.1, patience=2, min_lr=0.00001, verbose=1),
]

# Train the model
history = model2_1.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    callbacks=callbacks
)
```

```
callbacks = [
    EarlyStopping(patience=5, monitor='val_loss', verbose=1),
    ModelCheckpoint('MobileNetV2_condition2_69.h5', save_best_only=True, monitor='val_loss', mode='min')
    # ReduceLROnPlateau(factor=0.1, patience=2, min_lr=0.00001, verbose=1),
]

# Train the model
history = model2_2.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // BATCH_SIZE,
    callbacks=callbacks
)
```

```
# Evaluate the trained model on the test set (run this when already done training)
model1 = load_model('custom_CNN_condition.h5')
test_loss, test_accuracy = model1.evaluate(test_generator)

# # Evaluate the model on the test set
# test_loss, test_accuracy = model.evaluate(test_generator)

# Print the test loss and accuracy
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
# Extracting training and validation accuracy and loss
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']


loss = history.history['loss']
val_loss = history.history['val_loss']


# Calculating the number of epochs
epochs = range(1, len(accuracy) + 1)

# Plotting accuracy
plt.plot(epochs, accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plotting loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
# Evaluate the trained model on the test set (run this when already done training)
model2 = load_model('MobileNetV2_condition2_1.h5')
test_loss, test_accuracy = model2.evaluate(test_generator)

# # Evaluate the model on the test set
# test_loss, test_accuracy = model2_1.evaluate(test_generator)

# Print the test loss and accuracy
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
# Extracting training and validation accuracy and loss (model2_1)
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']


loss = history.history['loss']
val_loss = history.history['val_loss']


# Calculating the number of epochs
epochs = range(1, len(accuracy) + 1)

# Plotting accuracy
plt.plot(epochs, accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plotting loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
# Evaluate the trained model on the test set (run this when already done training)
model3 = load_model('MobileNetV2_condition2_2.h5')
test_loss, test_accuracy = model3.evaluate(test_generator)

# # Evaluate the model on the test set
# test_loss, test_accuracy = model2_2.evaluate(test_generator)

# Print the test loss and accuracy
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
# Extracting training and validation accuracy and loss (model2_2)
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']


loss = history.history['loss']
val_loss = history.history['val_loss']


# Calculating the number of epochs
epochs = range(1, len(accuracy) + 1)

# Plotting accuracy
plt.plot(epochs, accuracy, 'b', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'r', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

# Plotting loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
# Predict the label of the test_images
probs = model1.predict(test_generator)
pred = np.argmax(probs,axis=1)

# Map the label
labels = (train_generator.class_indices)
labels = dict((v,k) for k,v in labels.items()) # {0: 'fresh', 1: 'rotten'}
pred_labels = [labels[k] for k in pred]
```

```
# Display 15 random pictures from the dataset with their labels
random_index = np.random.randint(0, len(test_df) - 1, 15)
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(25, 15),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(test_df.Filepath.iloc[random_index[i]]))
    if test_df.Label.iloc[random_index[i]] == pred_labels[random_index[i]]:
        color = "green"
    else:
        color = "red"
    ax.set_title(f"Actual: {test_df.Label.iloc[random_index[i]]}\nPredicted: {pred_labels[random_index[i]]}", color=color)
plt.show()
plt.tight_layout()
```

```
# Get true labels from the test generator
actual = test_generator.classes

# Plot confusion matrix
cm = confusion_matrix(actual, pred)
plt.figure(figsize=(8, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix (Custom CNN)')
plt.show()

# Print classification report
class_labels = list(test_generator.class_indices.keys())
print(classification_report(actual, pred, target_names=class_labels))
```

```
# Predict the label of the test_images
probs2 = model2.predict(test_generator)
pred2 = np.argmax(probs2,axis=1)

# Map the label
labels2 = (train_generator.class_indices)
labels2 = dict((v,k) for k,v in labels2.items()) # {0: 'fresh', 1: 'rotten'}
pred_labels2 = [labels2[k] for k in pred2]
```

```
# Display 15 random pictures from the dataset with their labels
random_index = np.random.randint(0, len(test_df) - 1, 15)
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(25, 15),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(test_df.Filepath.iloc[random_index[i]]))
    if test_df.Label.iloc[random_index[i]] == pred_labels2[random_index[i]]:
        color = "green"
    else:
        color = "red"
    ax.set_title(f"Actual: {test_df.Label.iloc[random_index[i]]}\nPredicted: {pred_labels2[random_index[i]]}", color=color)
plt.show()
plt.tight_layout()
```

```
# Get true labels from the test generator
actual = test_generator.classes

# Plot confusion matrix
cm = confusion_matrix(actual, pred2)
plt.figure(figsize=(8, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix (MobileNet V2)')
plt.show()

# Print classification report
class_labels = list(test_generator.class_indices.keys())
print(classification_report(actual, pred2, target_names=class_labels))
```

```
# Predict the label of the test_images
probs3 = model3.predict(test_generator)
pred3 = np.argmax(probs3,axis=1)

# Map the label
labels3 = (train_generator.class_indices)
labels3 = dict((v,k) for k,v in labels3.items()) # {0: 'fresh', 1: 'rotten'}
pred_labels3 = [labels3[k] for k in pred3]
```

```
# Display 15 random pictures from the dataset with their labels
random_index = np.random.randint(0, len(test_df) - 1, 15)
fig, axes = plt.subplots(nrows=3, ncols=5, figsize=(25, 15),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(test_df.Filepath.iloc[random_index[i]]))
    if test_df.Label.iloc[random_index[i]] == pred_labels3[random_index[i]]:
        color = "green"
    else:
        color = "red"
    ax.set_title(f"Actual: {test_df.Label.iloc[random_index[i]]}\nPredicted: {pred_labels3[random_index[i]]}", color=color)
plt.show()
plt.tight_layout()
```

```
# Get true labels from the test generator
actual = test_generator.classes

# Plot confusion matrix
cm = confusion_matrix(actual, pred3)
plt.figure(figsize=(8, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix (Pre-trained MobileNet V2)')
plt.show()

# Print classification report
class_labels = list(test_generator.class_indices.keys())
print(classification_report(actual, pred3, target_names=class_labels))
```

```
# Load and preprocess the image
img_path = 'testing/t1.jpg'
img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size as needed
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
img_array /= 255.  # Normalize pixel values

# Predict the label
probs = model1.predict(img_array)
pred_label_idx = np.argmax(probs)
pred_label = labels[pred_label_idx]

# Display the image and predicted label
plt.imshow(img)
plt.title(f'Predicted Label: {pred_label}')
plt.axis('off')
plt.show()
```

```
# Load and preprocess the image
img_path = 'testing/t1.jpg'
img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size as needed
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
img_array /= 255.  # Normalize pixel values

# Predict the label
probs = model2.predict(img_array)
pred_label_idx = np.argmax(probs)
pred_label = labels[pred_label_idx]

# Display the image and predicted label
plt.imshow(img)
plt.title(f'Predicted Label: {pred_label}')
plt.axis('off')
plt.show()
```

```
# Load and preprocess the image
img_path = 'testing/t2.jpg'
img = image.load_img(img_path, target_size=(224, 224))  # Adjust target_size as needed
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
img_array /= 255.  # Normalize pixel values

# Predict the label
probs = model3.predict(img_array)
pred_label_idx = np.argmax(probs)
pred_label = labels[pred_label_idx]

# Display the image and predicted label
plt.imshow(img)
plt.title(f'Predicted Label: {pred_label}')
plt.axis('off')
plt.show()
```

```dart
import 'package:flutter/material.dart';
import 'package:image_picker/image_picker.dart';
import 'package:flutter_tflite/flutter_tflite.dart';
import 'dart:io';

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: HomePage(),
    ); // MaterialApp
  }
}

class HomePage extends StatefulWidget {
  const HomePage({super.key});

  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  File? _image;
  final picker = ImagePicker();
  List<String> labelsList = [];

  @override
  void initState() {
    super.initState();
  }
```

```dart
37
38  Future<void> loadModel(String modelPath, String labelsPath) async {
39    String? res = await Tflite.loadModel(
40      model: modelPath,
41      labels: labelsPath,
42    );
43    print("Model loaded: $res");
44  }
45
46  Future<void> unloadModel() async {
47    await Tflite.close();
48    print("Model unloaded");
49  }
50
51  Future<void> _pickImage() async {
52    final pickedFile = await picker.pickImage(source: ImageSource.gallery);
53    setState(() {
54      if (pickedFile != null) {
55        _image = File(pickedFile.path);
56        labelsList.clear();
57      }
58    });
59
60    if (_image != null) {
61      await _runModel1();
62    }
63  }
64
```

```dart
Future<void> _takePicture() async {
  final pickedFile = await picker.pickImage(source: ImageSource.camera);
  setState(() {
    if (pickedFile != null) {
      _image = File(pickedFile.path);
      labelsList.clear();
    }
  });

  if (_image != null) {
    await _runModel1();
  }
}

Future<void> _runModel(String modelPath, String labelsPath) async {
  if (_image == null) {
    _showSnackBar('Please select an image first');
    return;
  }

  await loadModel(modelPath, labelsPath);

  var recognitions = await Tflite.runModelOnImage(
    path: _image!.path,
    numResults: 1,
    threshold: 0.5,
    imageMean: 0,
    imageStd: 255,
  );
  print("Recognitions: $recognitions");

```

```
96      if (recognitions != null && recognitions.isNotEmpty) {
97        var label = recognitions[0]['label'];
98        setState(() {
99          if (labelsList.isEmpty || labelsList.last != label) {
100           if (labelsList.length < 2) {
101             labelsList.add(label);
102           } else {
103             labelsList[1] = label;
104           }
105         }
106       });
107       print("Labels List: $labelsList");
108     }
109
110     await unloadModel();
111   }
112
113   Future<void> _runModel1() async {
114     await _runModel("assets/MobileNetv2_type3.tflite", "assets/labels_type.txt");
115   }
116
117   Future<void> _runModel2() async {
118     await _runModel("assets/MobileNetV2_condition2_2.tflite", "assets/labels_condition.txt");
119   }
120
121   void _showSnackBar(String message) {
122     final scaffoldMessenger = ScaffoldMessenger.of(context);
123
124     // Remove any existing SnackBar before showing a new one
125     scaffoldMessenger.removeCurrentSnackBar();
126
```

```
127       // Show the new SnackBar
128       scaffoldMessenger.showSnackBar(
129         SnackBar(
130           content: Text(message),
131           duration: const Duration(milliseconds: 500), // Set the duration to 0.5 seconds
132         ), // SnackBar
133       );
134     }
135
136     void _showPicker(BuildContext context) {
137       showModalBottomSheet(
138         context: context,
139         builder: (BuildContext context) {
140           return SafeArea(
141             child: Wrap(
142               children: <Widget>[
143                 ListTile(
144                   leading: const Icon(Icons.photo_library),
145                   title: const Text('Pick from Gallery'),
146                   onTap: () {
147                     _pickImage();
148                     Navigator.of(context).pop();
149                   },
150                 ), // ListTile
151                 ListTile(
152                   leading: const Icon(Icons.photo_camera),
153                   title: const Text('Take a Picture'),
154                   onTap: () {
155                     _takePicture();
156                     Navigator.of(context).pop();
157                   },
158                 ), // ListTile
159               ], // <Widget>[]
160             ), // Wrap
161           ); // SafeArea
162         },
```

```
163       );
164     }
165
166     @override
167     Widget build(BuildContext context) {
168       return Scaffold(
169         appBar: AppBar(
170           title: const Text(
171             'Apple Types & Condition Classification',
172             style: TextStyle(
173               fontWeight: FontWeight.bold,
174               color: Colors.white,
175               fontSize: 20,
176             ), // TextStyle
177           ), // Text
178           backgroundColor: const Color.fromARGB(255, 70, 130, 180),
179         ), // AppBar
180         backgroundColor: const Color.fromARGB(255, 234, 234, 234),
181         body: Center(
182           child: Column(
183             mainAxisAlignment: MainAxisAlignment.start, // Align items to the top
184             children: [
185               const SizedBox(height: 50), // Add space from the top
186               GestureDetector(
187                 onTap: _image == null ? () => _showPicker(context) : null,
188                 child: Container(
189                   width: 300,
190                   height: 300,
191                   decoration: BoxDecoration(
192                     border: Border.all(color: Colors.grey, width: 2),
193                     borderRadius: BorderRadius.circular(10),
194                     boxShadow: const [
195                       BoxShadow(
```

```
                    color: Colors.black26,
                    offset: Offset(0, 2),
                    blurRadius: 4.0,
                  ), // BoxShadow
                ],
              ), // BoxDecoration
              child: ClipRRect(
                borderRadius: BorderRadius.circular(10),
                child: _image == null
                    ? const Center(child: Text('No image selected.', style: TextStyle(color: Colors.grey)))
                    : Image.file(
                        _image!,
                        fit: BoxFit.cover, // Fit the image within the container
                      ), // Image.file
              ), // ClipRRect
            ), // Container
          ), // GestureDetector
          const SizedBox(height: 15), // Add space between image and buttons
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              ElevatedButton(
                onPressed: () {
                  setState(() {
                    _image = null;
                    labelsList.clear();
                  });
                  _showSnackBar('Image removed');
                },
                style: ElevatedButton.styleFrom(
                  backgroundColor: const Color.fromARGB(255, 70, 130, 180),
                ),
                child: const Text(
                  'Remove Image',
                  style: TextStyle(
```

```
                    color: Color.fromARGB(255, 255, 255, 255),
                    fontWeight: FontWeight.bold,
                  ), // TextStyle
                ), // Text
              ), // ElevatedButton
              const SizedBox(width: 10),
              ElevatedButton(
                onPressed: _runModel2,
                style: ElevatedButton.styleFrom(
                  backgroundColor: const Color.fromARGB(255, 70, 130, 180),
                ),
                child: const Text(
                  'Classify',
                  style: TextStyle(
                    color: Color.fromARGB(255, 255, 255, 255),
                    fontWeight: FontWeight.bold,
                  ), // TextStyle
                ), // Text
              ), // ElevatedButton
            ],
          ), // Row
          const SizedBox(height: 20), // Add space below buttons
          if (labelsList.length == 2) ...[
            Card(
              color: labelsList[0] != "Not Apple"
                  ? const Color.fromARGB(255, 230, 240, 255)
                  : const Color.fromARGB(255, 251, 251, 197),
              margin: const EdgeInsets.symmetric(vertical: 10, horizontal: 25),
              child: ListTile(
                leading: Image.asset(
                  labelsList[0] == "Granny Smith"
                      ? 'assets/green-apple.png'
                      : labelsList[0] != "Not Apple"
                          ? 'assets/apple.png'
                          : 'assets/apple-unknown.png',
```

```
                    width: 24,
                    height: 24,
                  ), // Image.asset
                  title: Text(labelsList[0], style: const TextStyle(fontSize: 20)),
                ), // ListTile
              ), // Card
              if (labelsList[0] != "Not Apple")
                Card(
                  color: labelsList[1] == "fresh"
                      ? const Color.fromARGB(255, 222, 253, 222)
                      : const Color.fromARGB(255, 252, 190, 199),
                  margin: const EdgeInsets.symmetric(vertical: 10, horizontal: 25),
                  child: ListTile(
                    leading: Image.asset(
                      labelsList[1] == "fresh"
                          ? 'assets/fresh.png'
                          : 'assets/rotten.png',
                      width: 24,
                      height: 24,
                    ), // Image.asset
                    title: Text(labelsList[1], style: const TextStyle(fontSize: 20)),
                  ), // ListTile
                ), // Card
            ],
          ],
        ), // Column
      ), // Center
    ); // Scaffold
  }
}
```

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: Week 2**
Student Name & ID: Tin Vi Leed 21ACB04826	
Supervisor: Dr Muhammad Syaiful Amri Bin Suhaimi	
Project Title: Classification of types and conditions of an apple (fruit) using deep learning model for consumer assessment	

1. WORK DONE

Revise the work from FYP1, collect more dataset.

2. WORK TO BE DONE

Collect dataset for “not” apple class.

3. PROBLEMS ENCOUNTERED

Time consuming for collecting the images.

4. SELF EVALUATION OF THE PROGRESS

Keep the progress in timeline

Supervisor’s signature

Student’s signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: Week 4**
Student Name & ID: Tin Vi Leed 21ACB04826	
Supervisor: Dr Muhammad Syaiful Amri Bin Suhaimi	
Project Title: Classification of types and conditions of an apple (fruit) using deep learning model for consumer assessment	

1. WORK DONE

Collect images for "not apple" class, revise work from FYP1.

2. WORK TO BE DONE

Train model with new dataset added, study on mobile application implementation.

3. PROBLEMS ENCOUNTERED

N/A.

4. SELF EVALUATION OF THE PROGRESS

Everything is worked as expected.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: Week 6**
Student Name & ID: Tin Vi Leed 21ACB04826	
Supervisor: Dr Muhammad Syaiful Amri Bin Suhaimi	
Project Title: Classification of types and conditions of an apple (fruit) using deep learning model for consumer assessment	

1. WORK DONE

Select best performing model, study basics of Flutter Dart coding.

2. WORK TO BE DONE

Design UI of mobile app, deploy model to the app

3. PROBLEMS ENCOUNTERED

Learning and working on code are time consuming. Encountered error regarding Flutter dependency.

4. SELF EVALUATION OF THE PROGRESS

Spend too much time on researching and learning. Need to solve the dependency issue as soon as possible to keep on timeline.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: Week 8**
Student Name & ID: Tin Vi Leed 21ACB04826	
Supervisor: Dr Muhammad Syaiful Amri Bin Suhaimi	
Project Title: Classification of types and conditions of an apple (fruit) using deep learning model for consumer assessment	

1. WORK DONE

Flutter dependency issues solved, integrate single model into code.

2. WORK TO BE DONE

Look for method to load multiple models.

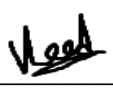
3. PROBLEMS ENCOUNTERED

Progress delayed due to model loading issues. Spent a lot of times to figure the problem and look for solution online.

4. SELF EVALUATION OF THE PROGRESS

Need to speed up progress to keep on timeline.

Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: Week 10**
Student Name & ID: Tin Vi Leed 21ACB04826	
Supervisor: Dr Muhammad Syaiful Amri Bin Suhaimi	
Project Title: Classification of types and conditions of an apple (fruit) using deep learning model for consumer assessment	

1. WORK DONE

Model loading issues solved, can load multiple models now. Both model for apple types and conditions are integrated into the code.

2. WORK TO BE DONE

Design UI of app. Mobile application testing.

3. PROBLEMS ENCOUNTERED

N/A

4. SELF EVALUATION OF THE PROGRESS

Even though a lot of issues encountered, it still managed to overcome and proceed to the tasks.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: Week 12**
Student Name & ID: Tin Vi Leed 21ACB04826	
Supervisor: Dr Muhammad Syaiful Amri Bin Suhaimi	
Project Title: Classification of types and conditions of an apple (fruit) using deep learning model for consumer assessment	

1. WORK DONE

UI of app is completely designed. Mobile application is tested and works normally.

2. WORK TO BE DONE

Documentation of FYP2.

3. PROBLEMS ENCOUNTERED

N/A

4. SELF EVALUATION OF THE PROGRESS

Everything is work as expected.

Supervisor's signature

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3S3	**Study week no.: Week 14**
Student Name & ID: Tin Vi Leed 21ACB04826	
Supervisor: Dr Muhammad Syaiful Amri Bin Suhaimi	
Project Title: Classification of types and conditions of an apple (fruit) using deep learning model for consumer assessment	

1. WORK DONE

Documentation of FYP2

2. WORK TO BE DONE

N/A

3. PROBLEMS ENCOUNTERED

N/A

4. SELF EVALUATION OF THE PROGRESS

All tasks are done and accomplished as expected.

Supervisor's signature

Student's signature

POSTER


UTAR
FACULTY OF INFORMATION COMMUNICATION AND TECHNOLOGY
Classification of Types and Conditions of an Apple (Fruit) using Deep Learning Model For Consumer Assessment
ABSTRACT
This project involved deep learning techniques and mobile technologies to implement a mobile application that will enable consumers to assess apple types and conditions quickly and accurately, thereby enhancing their shopping experiences and reducing time-consuming task of manual apple assessment.
OBJECTIVES
To develop a a robust deep learning model to classify apple types and conditions in real world scenarios.
To optimize for deep learning model deployment on mobile devices.
To enhance user experience by implementing a user-friendly interface application.
HOW TO USE?
1
Take images or upload image of an apple.
2
Types and conditions of apple will be displayed along with the uploaded image.
3
Select your preferred apple.
WHY USE THE PROPOSED SYSTEM?
1) User-friendly mobile application
2) Enhance shopping experiences with fast and accurate apple types and conditions classification.
3) Avoid unnecessary wastage due to wrong selection of apples.
Project Developer: Tin Vi Leed
Project Supervisor: Dr Muhammad Syaiful Amri Bin Suhaimi

PLAGIARISM CHECK RESULT

Turnitin Originality Report

Processed on: 04-Sep-2024 03:05 +08
ID: 2443741375
Word Count: 8234
Submitted: 4

21ACB04826_FYP2 (check plagiarism).pdf By Tin Vi Leed

Similarity Index	Similarity by Source	
9%	Internet Sources:	4%
	Publications:	5%
	Student Papers:	4%

2% match (publications)
Mehdi Ghayoumi. "Generative Adversarial Networks in Practice", CRC Press, 2023

1% match (student papers from 27-Apr-2023)
Submitted to Universiti Tunku Abdul Rahman on 2023-04-27

< 1% match (student papers from 02-May-2023)
Submitted to Universiti Tunku Abdul Rahman on 2023-05-02

< 1% match (student papers from 26-Apr-2024)
Submitted to Universiti Tunku Abdul Rahman on 2024-04-26

< 1% match (student papers from 27-Apr-2023)
Submitted to Universiti Tunku Abdul Rahman on 2023-04-27

< 1% match (Internet from 30-Mar-2023)
http://eprints.utar.edu.my/4715/1/fyp_IA_2022_CWS.pdf

< 1% match (Internet from 16-Sep-2023)
http://eprints.utar.edu.my/5526/1/fyp_IA_2023_ACD.pdf

< 1% match (Internet from 26-May-2024)
http://eprints.utar.edu.my/5513/1/fyp_IB_2023_CPM.pdf

< 1% match (Internet from 20-Jan-2024)
http://eprints.utar.edu.my/5508/1/fyp_CT_2023_TKH.pdf

< 1% match (student papers from 08-May-2023)
Submitted to Indiana University on 2023-05-08

< 1% match (student papers from 02-Jun-2024)
Submitted to Bahrain Polytechnic on 2024-06-02

< 1% match (student papers from 16-Apr-2024)
Submitted to Houston Community College on 2024-04-16

< 1% match (Internet from 15-Apr-2021)
https://espace.curtin.edu.au/bitstream/handle/20.500.11937/879/14464_Wang%2c%20Jiande%202003.pdf?isAllowed=y&sequence=2

< 1% match (publications)
C Kishor Kumar Reddy, P R Anisha, Samiya Khan, Marlia Mohd Hanafiah, Lavanya Pamulaparty, R Madana Mohana. "Sustainability in Industry 5.0 - Theory and Applications", CRC Press, 2024

< 1% match (student papers from 21-Aug-2022)
Submitted to Monash University on 2022-08-21

< 1% match (Internet from 01-Feb-2024)
https://agriculture-ataunipress.org/Content/files/sayilar/129/AUJAF_Jaunary_2024-26-35.pdf

< 1% match (student papers from 16-Oct-2023)
Submitted to University of North Texas on 2023-10-16

< 1% match (publications)
Parikshit N. Mahalle, Namrata N. Wasatkar, Gitanjali R. Shinde. "Data-Centric Artificial Intelligence for Multidisciplinary Applications", CRC Press, 2024

< 1% match (student papers from 10-Jun-2023)
Submitted to University of Cincinnati on 2023-06-10

< 1% match ()
Mulyani, Sri Hasta, Diqi, Mohammad, Salsabil, Husna Arwa. "GENERATIVE ADVERSARIAL NETWORKS FOR ANTERIOR CRUCIATE LIGAMENT INJURY DETECTION", Informatika, Universitas Jenderal Soedirman, 2024

< 1% match (student papers from 08-Aug-2024)
Submitted to National College of Ireland on 2024-08-08

< 1% match (Internet from 24-Mar-2024)
https://dev.thetechedvocate.org/how-to-install-android-studio-and-sdk-tools-in-windows/

< 1% match (student papers from 25-Aug-2023)
Submitted to University of Hull on 2023-08-25

< 1% match (student papers from 25-Jun-2024)
Submitted to unibuc on 2024-06-25

< 1% match (student papers from 28-Apr-2021)
Submitted to Segi University College on 2021-04-28

< 1% match (student papers from 16-Aug-2024)
Submitted to University of Hertfordshire on 2024-08-16

< 1% match (Internet from 11-Dec-2019)
https://aboutreact.com/installation-guide-for-visual-studio-code-code-editor/

< 1% match (Internet from 22-Mar-2024)
https://migrationletters.com/index.php/ml/article/download/8343/5372/21550

< 1% match (student papers from 02-Sep-2024)
Submitted to uwe on 2024-09-02

< 1% match (Internet from 02-Aug-2024)
https://www.wiseguyreports.com/reports/dry-shampoo-powder-market

< 1% match (student papers from 09-Feb-2024)
Submitted to Asia Pacific Instutute of Information Technology on 2024-02-09

< 1% match (student papers from 05-Jan-2024)
Submitted to University of Bedfordshire on 2024-01-05

< 1% match (Internet from 03-Dec-2019)
https://learnbatta.com/blog/python-introduction-to-programming-33/

< 1% match (Sovon Chakraborty, F.M. Javed Mehedi Shamrat, Md. Masum Billah, Md. Al Jubair, Md. Alauddin, Rumesh Ranjan. "Implementation of Deep Learning Methods to Identify Rotten Fruits", 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI), 2021)
Sovon Chakraborty, F.M. Javed Mehedi Shamrat, Md. Masum Billah, Md. Al Jubair, Md. Alauddin, Rumesh Ranjan. "Implementation of Deep Learning Methods to Identify Rotten Fruits", 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI), 2021

< 1% match (student papers from 09-May-2023)
Submitted to University of the Pacific on 2023-05-09

< 1% match (Kaiju Li, Chunhua Xiao. "PBFL: Communication-Efficient Federated Learning via Parameter Predicting", The Computer Journal, 2023)
Kaiju Li, Chunhua Xiao. "PBFL: Communication-Efficient Federated Learning via Parameter Predicting", The Computer Journal, 2023

< 1% match (Internet from 03-May-2024)
http://healthdocbox.com/Dental_Care/66308515-Sicat-surgical-guides-instructions-for-preparation-of-sicat-classicguide-and-sicat-optiguide.html

< 1% match (Internet from 04-Feb-2022)
https://link.springer.com/article/10.1007/s11998-021-00557-y?code=36b68030-4ccc-4642-9ab7-5c891675a832&error=cookies_not_supported

< 1% match ("Medical Image Computing and Computer Assisted Intervention – MICCAI 2018", Springer Nature America, Inc, 2018)
"Medical Image Computing and Computer Assisted Intervention – MICCAI 2018", Springer Nature America, Inc, 2018

< 1% match (publications)
H S Madhusudhan, Punit Gupta, Pradeep Singh Rawat. "Advanced Computing Techniques for Optimization in Cloud", CRC Press, 2024

< 1% match (publications)
John Brady, Alison Ebbage, Ruth Lunn. "Environmental Management in Organizations - The IEMA Handbook", Earthscan, 2013

< 1% match (student papers from 19-Aug-2023)
Submitted to Liverpool John Moores University on 2023-08-19

< 1% match (Md. Rakibul Islam, Md. Esraz-Ul-Zannat. "Remote sensing based investigation of coastal LULC dynamics in the coastal region of Bangladesh", Remote Sensing Applications: Society and Environment, 2023)
Md. Rakibul Islam, Md. Esraz-Ul-Zannat. "Remote sensing based investigation of coastal LULC dynamics in the coastal region of Bangladesh", Remote Sensing Applications: Society and Environment, 2023

< 1% match (publications)
Mike Tooley, Mike Tooley, Lloyd Dingle, Lloyd Dingle. "Higher National Engineering", Routledge, 2019

< 1% match ()
Junda, E, Malaga Chuquitaype, C, Ketsarin, C. "Interpretable machine learning models for the estimation of seismic drifts in CLT buildings", Journal of Building Engineering, 2023

< 1% match (Internet from 21-Feb-2024)
https://www.research.unipd.it/retrieve/e14fb26f-dca3-3de1-e053-1705fe0ac030/Cascini_Alessandro_tesi.pdf

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Tin Vi Leed
ID Number(s)	21ACB04826
Programme / Course	Bachelor of Computer Science (Honours)
Title of Final Year Project	CLASSIFICATION OF TYPES AND CONDITIONS OF AN APPLE (FRUIT) USING DEEP LEARNING MODEL FOR CONSUMER ASSESSMENT

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 9 % **Similarity by source** Internet Sources: 4 % Publications: 5 % Student Papers: 4 %	
Number of individual sources listed of more than 3% similarity: NA	

Parameters of originality required and limits approved by UTAR are as Follows:

(i) Overall similarity index is 20% and below, and

(ii) Matching of individual sources listed must be less than 3% each, and

(iii) Matching texts in continuous block must not exceed 8 words

Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Signature of Supervisor

Name: Muhammad Syaiful Amri bin Suhaimi

Date: 09/09/2024

Signature of Co-Supervisor

Name: _____

Date: _____

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	21ACB04826
Student Name	Tin Vi Leed
Supervisor Name	Dr Muhammad Syaiful Amri Bin Suhaimi

TICK (√)	DOCUMENT ITEMS Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
N/A	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)
Date: 26 August 2024