DES1

Fripe: Fruit Ripeness Detection App

FYP Final Report

By:

KIM Ji Hoo, KIM Jin Gee, KIM Taek Jung, YOON Jun Young

DES1

Advised By:

Dr. Desmond Yau-chat Tsoi

Submitted in partial fulfillment of the requirements of

COMP 4981 or CPEG 4901

in the

Department of Computer Science and Engineering

The Hong Kong University of Science and Technology

2020-2021

Date of Submission:

14 April 2021

Abstract

Ripeness is an important factor that affects the taste and nutrition of a fruit. Currently, most people measure ripeness by using their sensory skills to guess if the fruit is ripe enough. The accuracy of this method is dependent on one's experience and senses, which tend to be inconsistent. More accurate methods, such as using gas chromatographic or spectrometers, require extra tools which are inefficient and complicated. In this project, we implemented a mobile application that measures the ripeness of fruits using object detection and ripeness classification algorithms without any additional hardware.

Table of Contents:

Glossary

Color Representations

HSL - This stands for Hue, Saturation, and Luminance (or brightness). The HSL color space defines colors more naturally.

HSV - This stands for Hue, Saturation, Value is an alternative representation of the RGB color model. It models the way colors mix together[1].

L*a*b* - Also known as CIELAB, it expressed colors in three values: L* for lightness from black to white, a* from green to red, and b* from blue to yellow.

RGB - The most common type of color model and it stands for Red, Green, Blue. It represents color by mixing these colors together.

YCrCb - One of multiple color models that separate intensity from color information. ‘Y’ is the luma component and Cb and Cr are the blue-difference and re-difference chroma components.

(CIE)XYZ - The color space encompasses all color sensations that are visible to a person with average eyesight. This is why XYZ is a device-invariant representation of color.

1. Introduction

1.1 Overview

As more and more people are aiming to maintain a healthier lifestyle, fruits are becoming an important staple in many people's lives. While many people who grow their own fruits can pick their fruits when they are ripe, the evolution of technology and the system of mass production makes it difficult for fruits to be picked when they are ready to be eaten. Instead, fruits are picked before they are ripe, and then artificially ripened in mass quantities to make them ready when they are sent out to the market.

Although this increases efficiency and quantity, the quality of the fruits suffers. To be more precise, the artificial ripening process happens in large groups, making the ripeness of each fruit become inconsistent. This moves the responsibility to the people who are shopping for fruits to figure out the ripeness of each fruit. The solution that most people are using currently is hand-measuring a fruit's weight, viewing the color of the fruit's skin, and even tapping on the surface of the fruit to see what kind of sounds it makes. These methods are inaccurate and are based on human instincts and senses.

Although many technologies can calculate the ripeness of fruits, there are extremely few that can be used by consumers. One example is a gas chromatographic sensor that can detect ethylene levels in fruits to measure ripeness. Although they can be fairly accurate, it is implausible for any consumer to have one in their possession. Figure 1 shows the complexity of one of these devices.

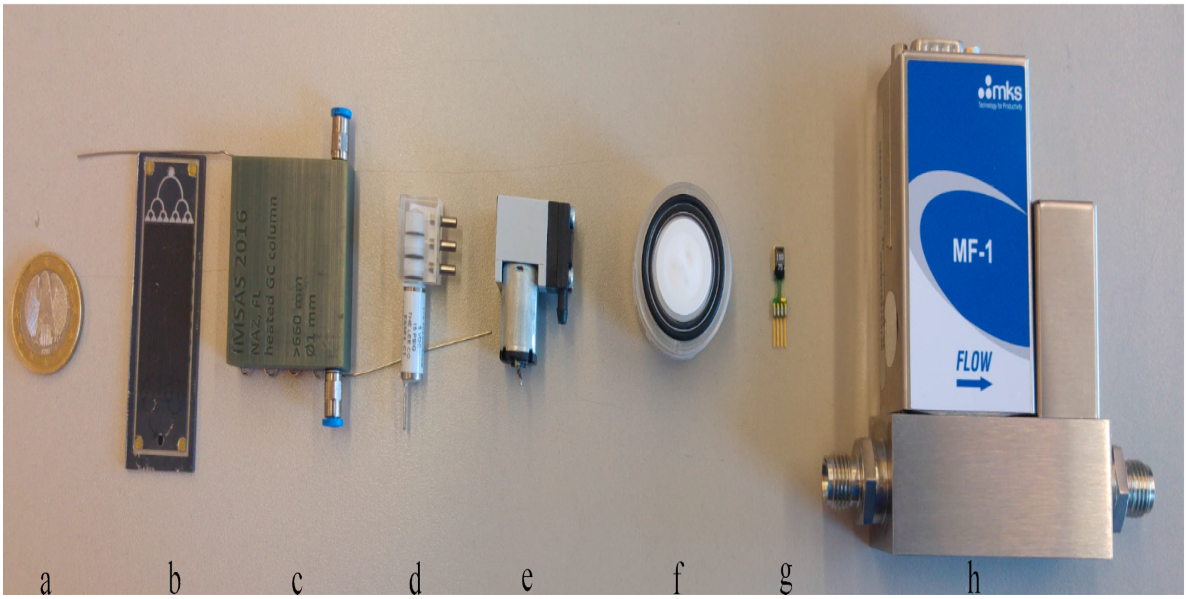


Figure 1 - Gas Chromatographic Sensor System[2]

Another available technology is a combination of a spectrometer and a smartphone that scans the physical surface of a fruit to measure ripeness[2]. Designed by a company called Clarifruit, this system uses the spectrometer to obtain data about the fruit's color and complexion. It also gathers more data about the fruit using the smartphone's camera. After combining these two data inputs, the data is sent to the cloud where it can be analyzed to give a result about the fruit's ripeness. Although this implementation is much more feasible for consumers to use than the gas chromatographic sensor, it also requires consumers to carry around another device with them. Carrying around the spectrometer whenever a customer goes shopping can be tedious and reduce functionality. Our application supplements the problems of the aforementioned technologies. It can be loaded into a cell phone that everyone carries, and a high accuracy is achieved using machine learning algorithms.

Here is a comparison between our application and the existing technologies stated above.

	Instinct	Gas Chromatographic sensor	Application with a third-party hardware	Our Application
Convenience How easy it is for consumer to use the method	5	1	3	4.5
Accessibility How easy it is for consumer to access the method	5	1	2	4.5
Accuracy How accurate can it measure the ripeness	1	5	4	4
Total Score	11	7	9	13

Table 1 - Comparison between current technology (1 = worst, 5 = best)

As shown in Table 1, it can be seen that in terms of convenience, accessibility, and accuracy, our application has a relatively good balance in the three areas compared to current existing methods. Consumers can easily use our application to measure the ripeness with high accuracy.

At this point, a question arises in our mind regarding the ripeness detection. If we can identify the fruit's features with our eyes, why do we need an AI application to measure the ripeness for us? The answer to this is that our eyes have physical limitations in perceiving color. More specifically, a camera can identify fifteen more shades of color than the human eye[3]. As can be seen in Figure 2, for a human, it seems that these two fruits are extremely similar in color. But an analysis of the color schemes shows that these two apples have very different colors. This small difference can make an impact on how ripe the fruit is, which is why we need machine learning to help us make these decisions.

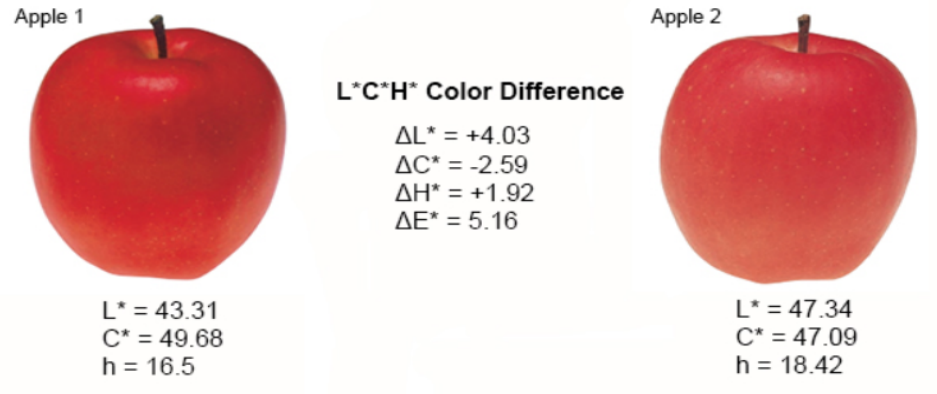


Figure 2 - A color difference in apples[4]

With the increase in computational power and the improvement of machine learning models, our team believes the method of determining fruit ripeness can be simplified even further. This means not having to think about large, expensive machines or carrying around unnecessary weight. Our project aims to minimize the functionality of determining fruit ripeness to our smartphones only.

1.2 Objectives

The primary objective of this project is to create a mobile application that measures the ripeness of fruits captured from a smartphone camera by adopting a deep neural network. This application can be separated into the following objectives.

Objective 1: Accurately identify fruits in real-time

For the first objective, there are two factors we need to consider: accuracy and speed of the detection. The model should provide correct information about the type of fruit and fast enough to support real-time detection on devices. To ensure this, only quality data should be used for training the model. Collected data should go through inspection procedure to make sure they are correctly annotated. YOLO and SSD were object detection models that we implemented since these models have been researched to have relatively fast inference speed and accuracy.

Objective 2: Precisely calculate the percentage of ripeness of fruits

For the second objective, deep neural network models should be able to precisely detect the ripeness of the detected fruits. More precisely, the models should be able to calculate the ripeness of the fruits with very similar colors differently, although they look the same in our eyes. In order to accomplish this objective, several factors should be considered to determine the ripeness: size, weight, color characteristic, and more. Thus, separate models will be constructed and trained for each type of fruit, with thousands of ripe and unripe fruit images.

Objective 3: Provide a user-friendly and intuitive interface with high user-experience

Our last objective is to minimize the complexity and maximize the usability of the application. The goal of the application is to improve people's lives, and this improvement cannot be achieved without constructing a suitable medium that people can use. No matter how efficient and accurate the object detection and ripeness classification models are, if the experience of the application is poor, users will not use the application. Therefore, it is important to make sure that users will have an enjoyable experience while using the mobile application. This includes designing a user-interface that is pleasing to the eye and a seamless experience without any disruptive behaviour or lag.

1.3 Literature Survey

Plenty of well-known machine learning technologies for object detection and ripeness classification were examined and compared to ensure that our models are feasible. Furthermore, numerous similar applications were evaluated to further improve the user interface, functionalities, and structures of our application.

1.3.1 Multivariate Analysis and Machine Learning for Ripeness Classification of Cape Gooseberry Fruits

Although the color is the primary characteristic we will be using to do image classification, there are also different types of color schemes that work best for each fruit. “Multivariate Analysis and Machine Learning for Ripeness Classification of Cape Gooseberry Fruits” is a paper that describes machine learning techniques for ripeness classification in Cape Gooseberry Fruits[5]. The paper mentions that for each fruit, the machine learning model must be different. This is because while some fruits rely on RGB coloration to determine ripeness, other fruits may rely on different color representation models to render the most accurate results. These include RGB, HSV, L*a*b*, HSI, VIS/NIR, FTIR, and more. Sometimes, even combinations of these color representation models produce the best results. Therefore, different machine learning models perform differently in regard to these color representation models. Using this knowledge to our advantage, we now know that we need to run multiple models for each fruit in order to ensure that the accuracy of our results will be maximized.

1.3.2 Clarifruit

Clarifruit is an application that is made to minimize the waste of fruits. Clarifruit claims that 45% of fruits are being wasted during the process of supply for various reasons, although the majority of them could have been acceptable[6]. The application is used by growers, marketing companies, wholesalers, and retailers to measure the quality of fruits to help them make the right decisions to reduce waste and maximize profits across the supply chain.

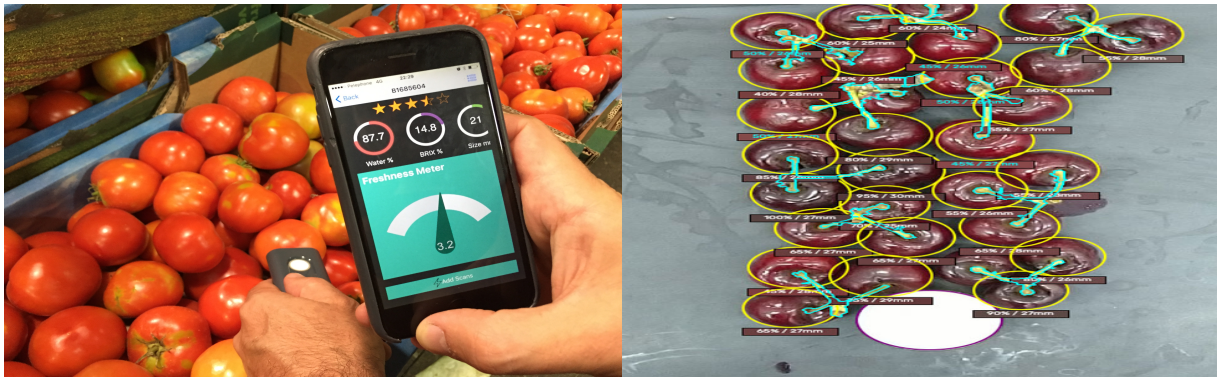


Figure 3 - Usage of Clarifruit and the outcome[7-8]

Clarifruit uses computer vision and machine learning technology to analyze the external attributes, which are color, stem color, and size. This is very similar to the technology that we used in our application. Furthermore, the application also measures internal attributes with third-party hardware, for example, Brix refractometer, durometer, penetrometer, PH device, and more[9]. These processes accurately and precisely check whether the fruits are acceptable to be sold.

This application is similar to our project in the aspect of examining the external attributes of the fruits. However, this application is made only for suppliers to check the quality of the fruits, and it needs various third-party hardware for measurements. Hence, the structure of the application is extremely complicated. Moreover, the detection time takes much longer than our application as it sends/fetches the data to/from the backend. Our application detects fruit and displays the ripeness in just a few seconds without any third-party hardware.

1.3.3 FoodTracker: A Real-time Food Detection Mobile Application by Deep Convolutional Neural Networks

Our application has two main functions; fruit detection, and ripeness detection. FoodTracker is similar to our application in the aspect of fruit detection. It uses deep learning technology to detect numerous types of food in real-time by deploying the trained model in the application. It also displays nutrition facts in the user interface, which inspires us to implement similar features in our application.

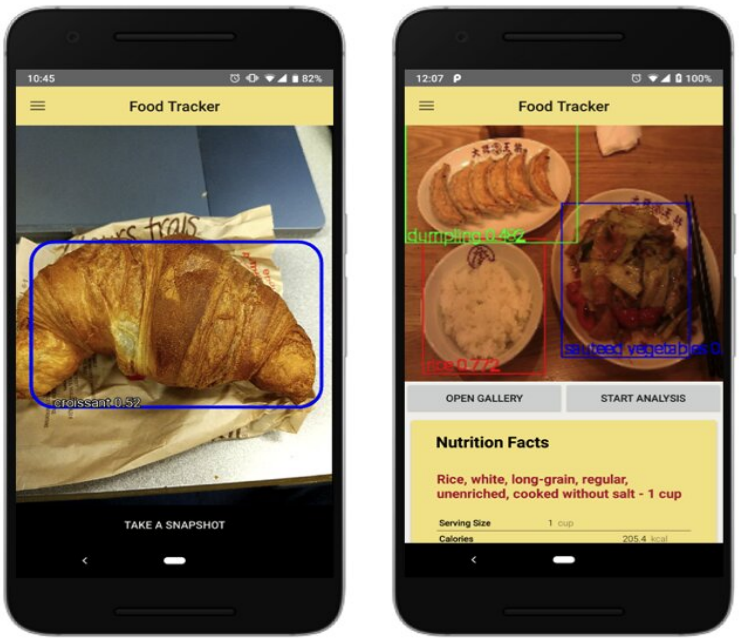


Figure 4 - Prototype of FoodTracker[10]

FoodTracker has very limited functionalities; it only detects the type of food and the corresponding nutrition facts. Furthermore, the nutrition values heavily depend on the specific types of food. For instance, even though brown rice, grain rice, and white rice have different nutrition values, this application categorizes all of them as “rice” with the same nutrition facts. In other words, although the application detects food with 100% accuracy, the nutrition facts are not accurate.

Our application not only detects the types of fruit but also detects the ripeness of the fruits. This resolves the limited functionalities of FoodTracker by maximizing the usability of the application. Moreover, we provide the possible consequences of consuming the detected unripe fruit and information about the detected fruit instead of nutrition data to resolve the inaccuracy of the nutrition facts.

1.4 Feasibility study

The primary concern related to our application is whether the performance of our machine learning algorithms is adequate enough to achieve our objectives. Thus, some tests were conducted to ensure the feasibility of our application. There are three main aspects to be inspected:

1. Accuracy of the object detection model
2. Speed of the object detection model
3. Accuracy of ripeness classification model

1.4.1 Accuracy of the object detection model

Some fruits have similar shapes and colors, which makes them hard to distinguish. Thus, there were some concerns about the accuracy of the object detection model. To test whether object detection can detect correctly, we have trained the YOLO-fastest model with a custom fruit dataset consisting of peach, tomato, and apple. In total, around 3,000 images were used to test the YOLO-fastest model.

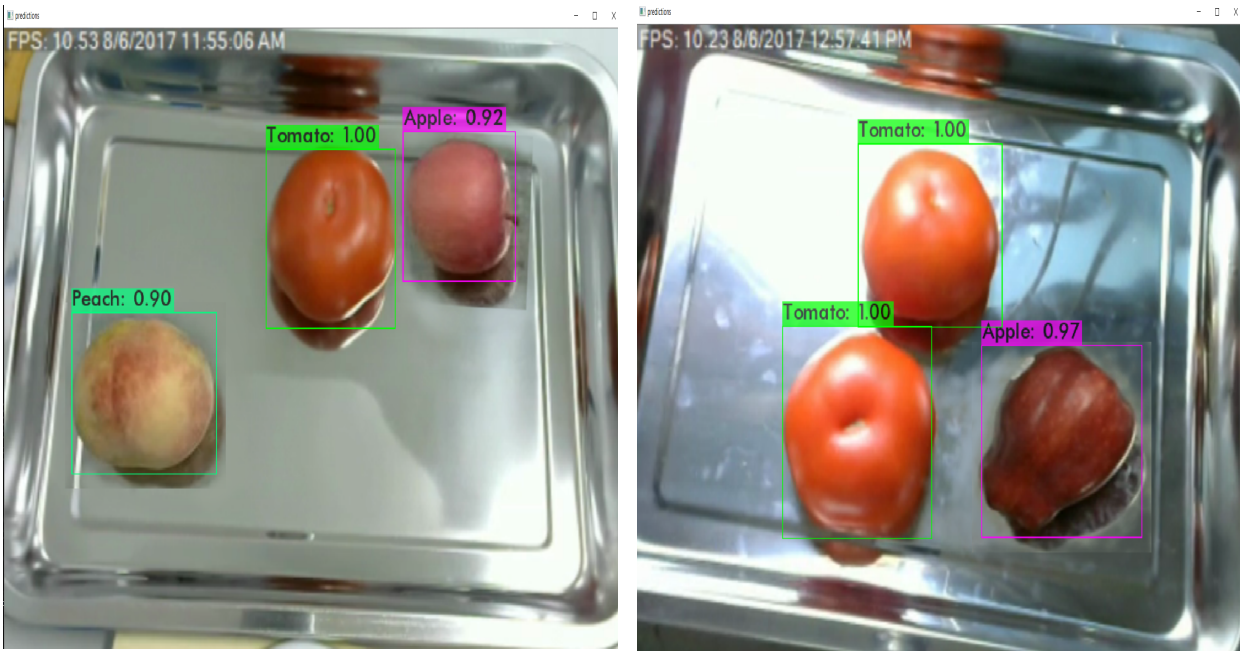


Figure 5 - Object detection feasibility test of different type of fruits[23]

The object detection model was able to detect different types of fruit correctly. As shown in Figure 5, YOLO-fastest was able to detect peach, tomato and apple correctly with high confidence rates.

1.4.2 Speed of object detection

Since our objective is to build applications that can support real-time detection on devices, our object detection model should have adequate speed. Anything below ten frames per second is difficult to use, so the cutline has been set to ten frames per second[11]. We have tested two methods for testing object detection on mobile devices, OpenCV modules, and tflite.

OpenCV is an open-source library for computer vision and machine learning[12]. By importing its DNN modules on android studio, we have built simple applications that can perform object detection[13]. Using this application, we have checked the speed of YOLOv4, YOLOv4-tiny, and YOLO-fastest.

	YOLOv4	YOLOv4-tiny	YOLO-fastest
Frames per Second (FPS)	0.4	4.7	4.5

Table 2 - Speed of object detection model using OpenCV dnn (tested on Galaxy s10)

From Table 2, it is known that by using the OpenCV dnn module, it was not able to detect fast enough to perform real-time detection.

Another method was using TensorFlow Lite. TensorFlow Lite is a tool that helps TensorFlow models to be run on mobile devices. It has a property of optimization which enhances the speed and energy efficiency of the model running on mobile devices[14]. Using TensorFlow github repository[15], we have built a TFLite object detection application.

	YOLOv4	YOLOv4-tiny	YOLO-fastest	SSD-mobilenet_v2_coco	SSD-mobilenet_v1_coco
Frames per Second (FPS)	0.7	5.3	24.3	11.3	10.2

Table 3 - Speed of object detection model using TFlite (tested on Galaxy s10)

From Table 3, we have found some models that have reached a speed above 10 FPS. The performance for YOLOv4 and YOLOv4-tiny has not improved much, where speed of YOLO-fastest has improved by a huge factor. Since there were three models that had the performance of higher than 10 FPS, it is indeed possible to perform real-time object detection on mobile devices.

1.4.3 Accuracy of ripeness classification model

Model's ability in selecting necessary features and the availability of color representations are considered to be important in achieving high accuracies.

1.4.3.1 Feature Extraction

Although there is some research on fruit ripeness classification models, we could not be sure whether the models can actually catch the important features of fruits. If the models cannot extract features that are significant in determining the ripeness, we are not able to get accurate results. To check the feasibility of the ripeness model, we implemented a CNN model to visualize feature/activation maps. The feature maps show the results of applying filters to previous layers. This can give some understanding of what features the models detect.

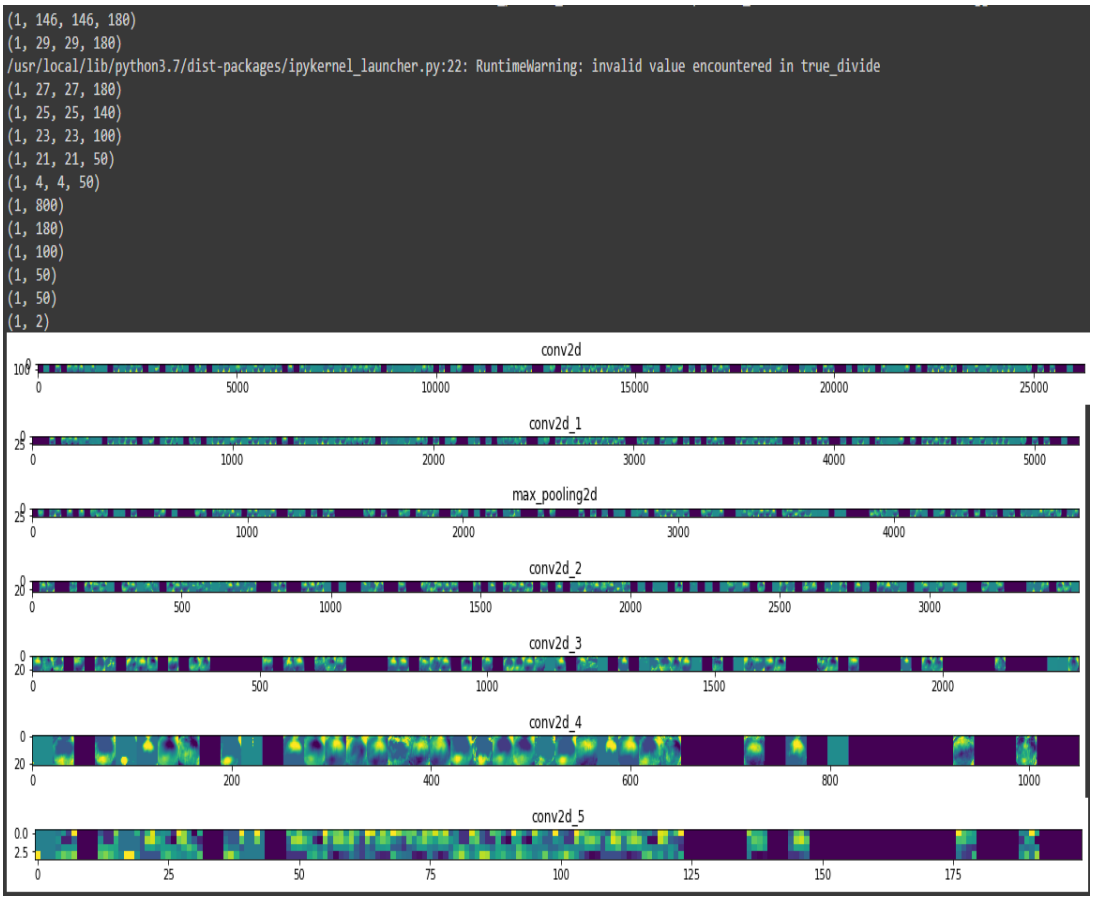


Figure 6 - Feature maps of CNN model

From the above feature maps (Figure 6), we can see that the model puts its importance on the shape and the color of the fruit image. The shape and the color are indeed the most necessary features that we have to look at in determining the type and ripeness of fruits. As a result, we believe if sufficient amounts of data are involved in training, we are able to build ripeness models with reasonable accuracies.

1.4.3.2 Color Representation

Another concern is related to the color representation. Color plays the biggest role in building a ripeness model. As ***1.3.1 Multivariate Analysis and Machine Learning for Ripeness Classification of Cape Gooseberry Fruits*** stated, different color representations can affect the accuracies of the models. Fortunately, the cvtColor() function from the cv2 library allows us to change the representation of images.

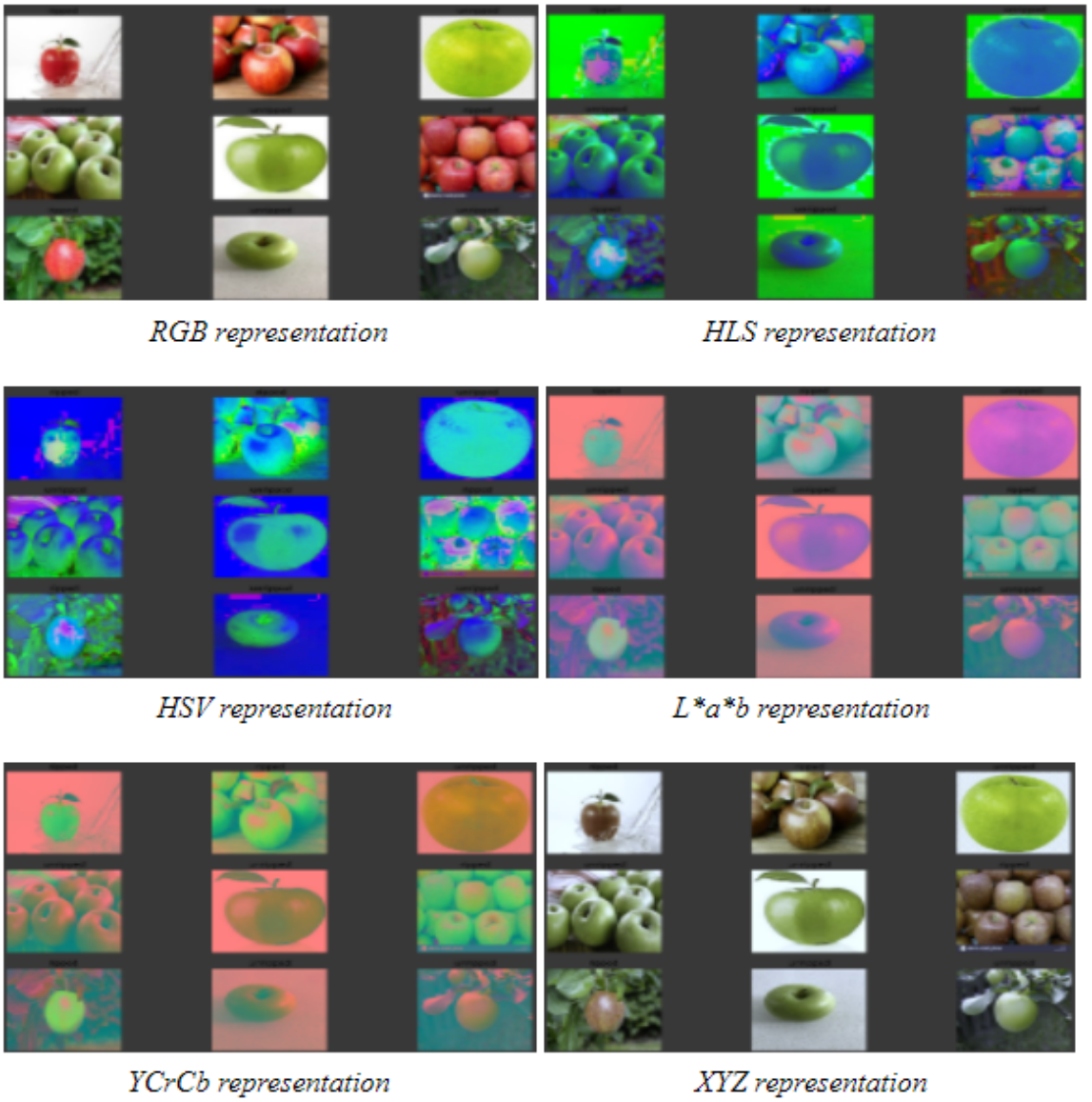


Figure 7 - Color representations of images

2. Methodology

2.1 Design

2.1.1 System Design

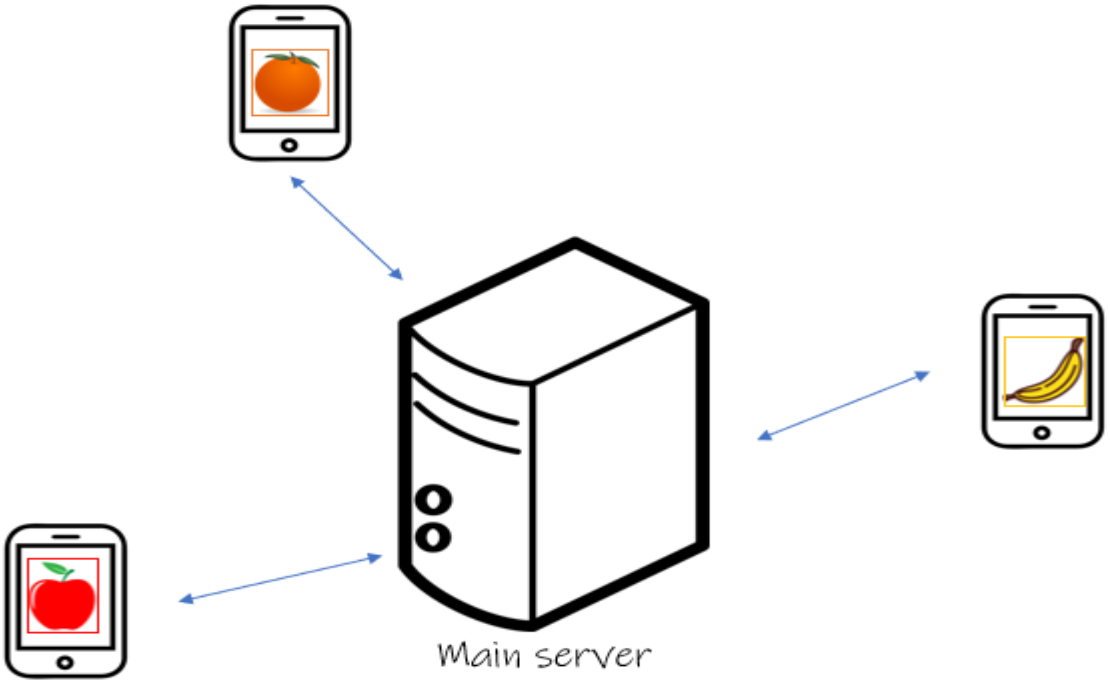


Figure 8 - Platform diagram

Our application consists of two main parts: the mobile application and the server. The mobile application can be further divided into object detection models and user experience. The object detection will be loaded directly into the mobile application so that it can perform object detection in real-time using the device's camera. The user interface will assemble all the components to create a wholesome and functional experience for the users who use our application. On the other hand, the fruit ripeness model will be deployed to the server so that it can be executed by the mobile application upon request. Figure 8 above shows a brief outline of the system, and Table 4 below lays out the specifications of each component.

Component	Function
Mobile Device	Object detection Loads the object detection model into the mobile application so that it can make predictions in real-time on the images that are captured by the device's camera.
	User interface and experience Shows the types of fruits that are being detected by the devices' camera and each fruit will have a bounding box surrounding it. Let the user choose which type of fruit he or she would like to find the ripeness level of, and show the ripeness when the user presses on one of the bounding boxes. Provides various beneficial information related to the selected fruit such as health benefits and harmful effects according to the nutrition facts.
Server	Deploy ripeness detection model to a server using Flask, and access the deployed models with REST API from and to the mobile devices.

Table 4 - Table of components included in the system design and their respective functions

2.1.2 Design choice and rationale

2.1.2.1 Running object detection on-device

Object detection is a deep neural network consisting of a large number of convolutional layers that require high computational power. Since the computational power of mobile devices is not as powerful as on the server's, many mobile applications that deal with deep neural network furnish servers to remotely execute the model. However, due to the characteristics of our application, deploying an object detection model on a server seemed unsuitable because of a few reasons explained below.

The first reason is that the expected usage of our application is concentrated on a certain time period and day. Many people go shopping after work at around 6pm - 9pm or during the weekends. A large number of users practicing the application at the same time might overwhelm the server, resulting in server traffic or even malfunction. Executing on-device object detection distributes the centralized computation on the server which resolves issues during rush hour.

Another reason is the delay caused by the network transfer. To run an object detection model on a server, the user's device has to transfer a stream of images to the server. These images will be processed sequentially and returned to the device with coordinates of the instances. Due to the round trip time, executing an object detection model on a server might not be fast enough to support on a real-time basis.

Based on these limitations, we have decided that it is more suitable to run object detection on devices.

2.1.2.2 Uploading ripeness models on the server

When a single fruit is chosen by the user, a cropped image of the fruit needs to be sent to one of the ripeness models, depending on the type of the fruit. We initially planned to put our ripeness models on-device so we can eliminate delays in the connection between the application and server; however, the size of the models came out to be too large to be stored in the application, creating long delays within the app and increasing battery consumption.

To solve this issue, we stored our models on a server. Whenever a user selects a fruit, the cropped image will be sent to the server with the corresponding model and the ripeness result will be sent back to the application. In this manner, we can show the ripeness percentage more efficiently.

2.1.2.3 Choice of Fruits

There were two main factors we considered when choosing the types of fruits: popularity of the fruits and accessibility of the fruit data.

The popularity of the fruits is the aspect we acknowledge to choose the fruit. Our target user is a consumer who is wondering about the ripeness of a fruit in the market. Therefore, we have chosen fruits that can be easily accessed by a wide range of users. Our research of the target market revealed that apples, bananas, grapes, lemons, mangos, oranges, peach, strawberries, tomatoes, and watermelons were the most common fruits sought after in the market[16-20]. In addition, our team members did field research to determine if the fruits were present in the market. After confirming their presence, we decided to go start with these ten fruit candidates.

Another aspect we had to consider was whether we were able to attain unripe fruit data. Since our project is to calculate fruit ripeness, numerous unripe fruit images have to be collected. Even though we have confirmed their presence in the market, it was difficult to attain an unripe image of all ten fruits mentioned above. Thus, we have narrowed it down to four fruits which are apple, mango, orange, and tomato due to difficulties in acquiring the data.

2.1.2.4 Returning ripeness of the fruit

Object detection can identify object instances in the image frame. However, it cannot identify if the object instance from the previous frame and the current frame are indeed equivalent. Thus, the ripeness value will be valid only on a single frame.

The method we have first taken into consideration was implementing a tracking algorithm to associate instances on a different frame. This algorithm finds the optimal matches by calculating the overlapping area of the predicted position and the actual position in the current frame. However, if n instances are in the image, it requires $n * n$ computational power to track each instance. This process requires high computational power which might reduce the performance (speed) of our application. Also, if the detection is not fast enough compared to the movement of the object, tracking will fail to associate instances since there will be no overlapping.

Due to technical issues related to tracking, we have come up with a simple solution, freezing the screen when calculating the ripeness. Since the screen is frozen, we are able to correctly return ripeness score to the corresponding fruit instance.

2.1.3 Dataflow

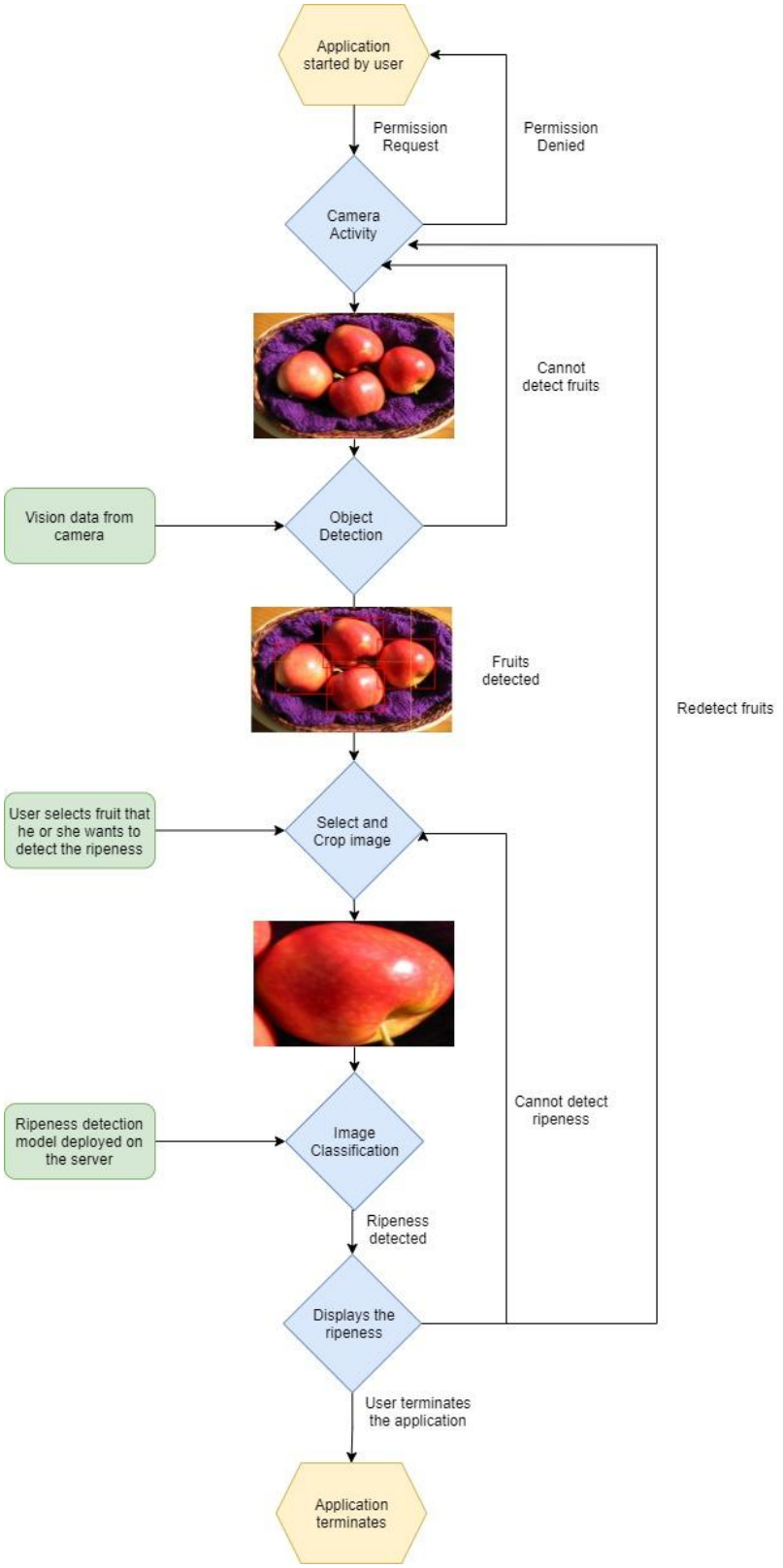


Figure 9 - Detailed data flow between components of the project

Figure 9 depicts more detailed information on the data flow of our fruit ripeness detection application. Accessing a camera requires permission from the user. After granting permission, the application will start the camera to start receiving input. The object detection model, which is loaded in the application, will detect fruits and create bounding boxes around fruit instances. When a user presses one of the bounding boxes, the application will crop the image and send it to the server. The server has various image classification models and their weights. Depending on the type of fruit detected, the cropped image will be fed into the respective image classification model. The ripeness result will be sent back to the mobile application and will display the ripeness of the selected fruit. This process continues until the user terminates the application.

2.2 Implementation

Based on our system design, our application can be divided into four sections: Fruit detection model, server implementation, fruit ripeness classification, and mobile application development.

2.2.1 Fruit detection model

As explained in our first objective, our application requires us to correctly identify types of fruits. We have explored various object detection models to find the most appropriate model to run on mobile devices. You Only Look Once (YOLO) and Single Shot Detector (SSD) use a fully convolutional approach that requires a single step to detect objects. These types of models are known to have a faster speed which is more suitable to be implemented on mobile applications than other region-based models such as faster-RCNN. Thus, we have decided to implement YOLO or SSD models.

2.2.1.1 Data collection

To train the object detection model, a sufficient number of images are required. Since data has a huge impact on the performance of the model, data has been collected attentively. There are few factors taken into consideration.

First, data was collected from various sources. Since training with similar data can cause models to become biased, the accuracy of the model might vary depending on the environment. Thus, the dataset was collected in three ways: Google images[21], Kaggle datasets[22-23], and video of fruits (YouTube videos[24-28] and fruit videos taken by us in the market).

Second, numerous non-fruit images were collected. Most fruits have a rounded shape and specific color (depending on the type of fruit). Thus, when training an object detection model with only fruit images, the model might only rely on color and shape of the object. This results in a model to false detect any object that is round, or has the same color as the fruit. To reduce false detection, non-fruit images have to be included in the training dataset to let models learn features of non-fruit objects.

Lastly, we also collected abnormal types of data such as unripped, over-ripped, and rotted fruit images. Our object detection model should detect fruits at any ripeness stage. However, without these abnormal data, the model will only detect fruits that are adequately ripped. Thus, various unripe fruit images were collected by image crawling from google (see *Appendix 6.1*).

In total, 28,160 images were collected for the object detection model.

2.2.1.2 Data preparation

We have trained our object detection model with images that have one or more objects. Each image file has a corresponding label file that contains the class and the location of the object presented in the image. We have used two annotation tools to generate a custom dataset for our application.

The first tool is CVAT (Computer Vision Annotation Tool)[29] which is a web-based annotation tool. It allows multiple users to access the data online, which helps to efficiently manage the dataset. Also, both image type and video type (converted to an image) datasets can be uploaded to the CVAT server to be annotated. However, due to the latency caused by network delay, we have also adopted an offline annotation application named LabelImg[30]. LabelImg is a Python-based graphical image annotation application with a Qt-based graphical interface. We used it during the inspection and correction of the dataset.

We have annotated images in bounding box format, which is drawing the smallest square that fits the object instance. Labels were extracted in the YOLO format and VOC format due to different requirements for each model. Each format and conversion code can be found in *Appendix 6.2*. Labels were also extracted as XML files, which is the primary data format required to train the SSD model. The screenshot of the annotation process can also be found in *Appendix 6.2*.

After the annotation, we have applied data augmentation methods to improve accuracy (see Appendix 6.3).

2.2.1.3 Object detection

YOLO and SSD models pursue different procedures in training due to their distinct frameworks and structures.

YOLO - Darknet

Darknet is an open-source neural network framework that allows training and testing YOLO object detection models[31]. Detailed installation procedure for Darknet framework will be explained in *Appendix 6.4*. Before training, labels for training Darknet were generated in YOLO format and placed in the same directory as the image files.

To train Darknet, it is required to have three components: data file, cfg file, and weights file. Data file is information about the dataset, such as the number of classes, name of each class, and path for the training dataset. Cfg file contains the architecture of the model and training parameters. The weights file is the parameter of the model. Due to changes in the number of classes, name of each class, and the training dataset, some adjustments were made: data file was created from scratch, model architecture was edited to detect 4 classes, anchors were recalculated, and batch size and the number of iteration were edited because of our computer specification. We used COCO pre-trained weights to reduce the training time.

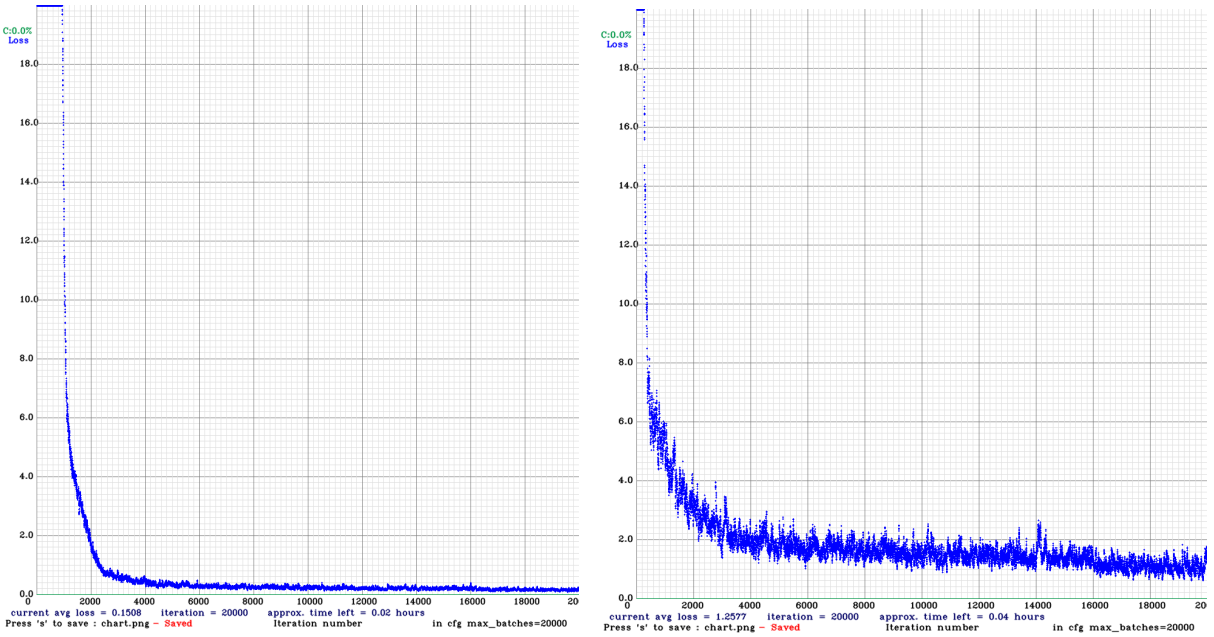


Figure 10 - Loss graph of YOLO-fastest and YOLOv4

Since the object detection model has to run on mobile devices, we have tested several YOLO models to find the most feasible one. The three YOLO models that have been tested are YOLOv4, YOLOv4-tiny, and YOLO-fastest[32]. The YOLOv4 model consists of 161 layers including 3 yolo layers. The size of the weights file for this model was 250,205KB, making it the largest. The YOLOv4-tiny model has 37 layers with only 2 yolo layers. Since the number of layers decreased by a huge factor, the size of the weights file decreased to 23,683KB. Lastly, YOLO-fastest contains 2 yolo layers and has a total of 125 layers. This model had the smallest weights file of 1,207KB. Even though it had more layers than YOLOv4-tiny, due to having a smaller filter size for each model, the weights file size was significantly reduced.

Mobilenet-SSD - Python

The mobilenet-ssd model is a Single-Shot Detection Network that can perform object detection on mobile devices[33]. Before training, we have to extract XML files from images as described in section ***2.2.1.2***. We trained the model with Google Colabotary[34] because it already has pre-installed Python packages.

To train SSD models with TensorFlow, we need to convert the data into a TensorFlow friendly format: TFRecord file. The TFRecord file is a converted version of the labeled data into a sequence of binary strings[35]. Although TFRecord files have very low readability, it saves a lot of space which helps in optimizing the object detection model, making it smaller and faster than other files such as XML or JSON. To generate TFRecord files, we need three components: CSV files, images, and a label file. The CSV files can be produced by concatenating the XML files that were extracted from the LabelImg software. Two CSV files had to be generated to split the dataset into a test and train dataset (we used 80 percent of the data for training, and the remaining 20 percent for testing). The Python code for concatenating XML files into CSV files is described in *Appendix 6.5.1*. The link used to generate TFRecord files is listed in *Appendix 6.5.2*.

Building the object detection model from scratch is an unnecessarily time-consuming job. Similar to the YOLO model, we have used a pre-trained model trained on a COCO dataset that was provided by TensorFlow developers[36]. The comparison of the speed and mAP of the pre-trained models will be explained in ***2.2.1.3 Evaluation of models***. The configurations of the model are not adjusted because the model hyperparameters were already chosen carefully by the TensorFlow object detection developers[37]. The only configurations we had to change were the batch size, number of classes, labelmap file path, and TFRecord file path. Since the estimation of the maximum batch size has an equation as follows,

Max batch size = (available GPU memory bytes / 4) / (size of tensors + trainable parameters)

we set the batch size to 12, which is reasonable to be implemented on modern devices. We also set the number of classes to 4, and the file paths were correctly configured according to the structure of our directories. The code for the implementation of the transfer learning on the ssd_mobilenet_v2_coco model is described in *Appendix 6.5.3*.

After the training is done, checkpoint files are created. TensorFlow checkpoint files contain the exact value of all parameters used by the model. However, they do not contain any description of the computation defined by the model, and thus we had to convert the checkpoint files into a protobuf format. The conversion code of the checkpoint files into the protobuf file is described in *Appendix 6.5.4*. The protobuf file includes a serialized description of the computation defined by the model, as well as the parameter values[38]. The generated protobuf file has a size of 18,875KB, and this file should be converted to tflite format which will be described in the ***2.2.1.5 Model Conversion*** section.

2.2.1.4 Evaluation of models

To evaluate the performance of the object detection model, we must consider both accuracy and speed. Mean Average Precision[39] and Frame per Second will be compared through models.

To calculate mAP of the model, the average precision of each class must be calculated. By running the test dataset on our trained model, we can obtain the prediction which will be compared with the ground truth to form a confusion matrix. For object detection, the true positive is when the prediction has the correct class and has IOU (intersection over union) to be above the threshold.

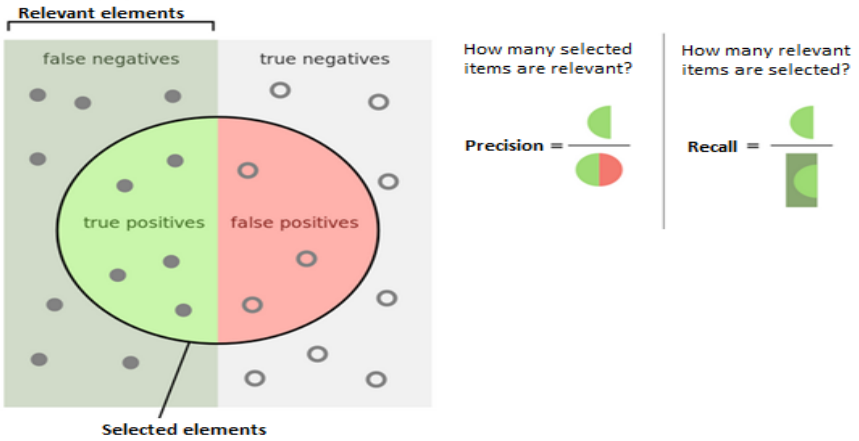


Figure 11 - Diagram of confusion matrix and explanation of Precision and Recall[40]

By using a confusion matrix, precision and recall can be calculated to form a precision-recall graph. The average precision of each class can be calculated by measuring the area under the curve, where mAP is the average of APs.

Frame per second represents how many image frames the object detection model processed in a second. By executing the same video footage on each object detection model, the FPS was calculated.

	mAP	FPS
YOLOv4	96.32%	21.3
YOLOv4-tiny	82.17%	69.2
YOLO-fastest	95.41%	63.4
SSD-mobilenet_v2_coco	88.43%	55.8
SSD-mobilenet_v1_coco	90.23%	45.4

Table 5 - mAP and FPS result of each model (GTX1070)

As shown in Table 5, mAP of the YOLO-fastest model was 0.91 lower than YOLOv4 (the most accurate model) and processed 5.8 frames less than YOLOv4-tiny (the fastest model). Thus, we decided that YOLO-fastest was the most suitable model for our project.

2.2.1.5 Model Conversion

Due to the mobile application development tools we are adopting, object detection models had to be converted into TensorFlow-Lite and and TensorFlow.js which can be easily transformed from the TensorFlow model. Since the YOLO model uses the Darknet framework, it was first converted to TensorFlow Python. Detailed information on conversion is in *Appendix 6.6.*

2.2.2 Fruit ripeness model

2.2.2.1 Data Collection & Preparation

Similar to the object detection model, sufficient amounts of image data are required to train fruit ripeness models and to get high accuracies. The same image dataset has been used as the detection model; collected from Google images, Kaggle datasets, and videos of fruit.

Since classification models can only identify one object per image, we have cropped images to reduce unwanted attributes such as backgrounds, leaves, stems, etc. By applying this augmentation method, it prevents models from learning undesirable features increasing accuracy (see *Appendix 6.7*).

To train a ripeness model or a classification model, we have to first classify images. For example, all the apple images have to be segregated and labeled as ripe and unripe so the machine can recognize which image belongs to which class or type. All different types of fruits went through this preprocessing to create distinct ripeness models.

2.2.2.2 Model Architecture Selection

Different machine learning models such as CNN, ANN, SVM, and LDA have varying architectures or processes in using neural networks and training the image data. We tried to implement all of these models to find out the best fitting model for each type of fruit. However, we encountered a problem. Since our application is running in real-time, the latency between the application and the server, where the ripeness models are stored, had to be minimized. Thus, the TensorFlow Serving system, which is very flexible and provides high performance, is used to connect between the application and the models. Although TensorFlow serving can accept other models, it can most easily accept TensorFlow models. Since SVM and LDA models are supported by the scikit-learn library but not TensorFlow Keras, we were not able to use those models for our application.

ANN and CNN models can be both built using the TensorFlow Keras library and we have conducted several tests to examine these two models. The major difference between these models is that, as shown in Figure 12, each neuron is connected to every other neuron for the ANN model, whereas only the last layer is fully connected for the CNN model[41]. Since all the neurons are connected to each other in ANN, the model easily leads to overfitting as the processor has to process large amounts of weights for each layer. To prevent overfitting, we have to resize the training images smaller so that the ANN model can handle the weights; however, this may lead to low accuracy as resolutions of the images will be decreased as the image sizes become smaller. Then, it becomes hard for the model to extract features from the images. The CNN model, on the other hand, only deals with a few weights for each layer. It goes through multiple layers to extract different features and only in the last layer it fully connects all the neurons. Therefore, it does not lead to overfitting like ANN.

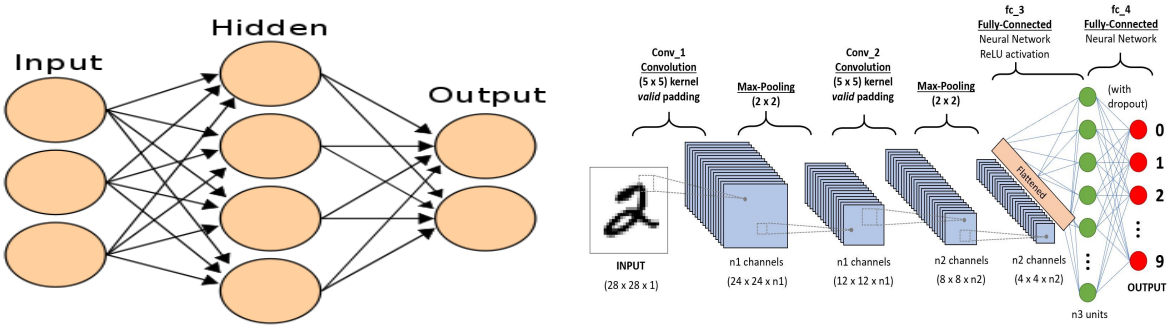


Figure 12 - Left shows ANN model and Right shows CNN model[42-43]

All of the fruits that we tried brought better performances with the CNN model. The CNN is known to have a better performance in analyzing images since it keeps spatial interaction between the pixels. We believe the CNN model is the most fittable model for our application both in terms of compatibility and accuracy.

Layer (type)	Output Shape	Param #
conv2d_78 (Conv2D)	(None, 148, 148, 200)	5600
conv2d_79 (Conv2D)	(None, 146, 146, 180)	324180
max_pooling2d_26 (MaxPooling	(None, 29, 29, 180)	0
conv2d_80 (Conv2D)	(None, 27, 27, 180)	291780
conv2d_81 (Conv2D)	(None, 25, 25, 140)	226940
conv2d_82 (Conv2D)	(None, 23, 23, 100)	126100
conv2d_83 (Conv2D)	(None, 21, 21, 50)	45050
max_pooling2d_27 (MaxPooling	(None, 4, 4, 50)	0
flatten_13 (Flatten)	(None, 800)	0
dense_51 (Dense)	(None, 180)	144180
dense_52 (Dense)	(None, 100)	18100
dense_53 (Dense)	(None, 50)	5050
dropout_13 (Dropout)	(None, 50)	0
dense_54 (Dense)	(None, 2)	102

Total params: 1,187,082
Trainable params: 1,187,082
Non-trainable params: 0

Figure 13 - Architecture of CNN model { image size:[150,150,3], classes:ripped and unripe}

2.2.2.3 Color Representation

Besides the selection of models, we also considered the color representations. When classifying an image through machine learning, the color is only and the most important factor that affects the results. This is because all the shapes and textures of fruits are represented as colors in images. Therefore different color representations can lead to various results or accuracies.

RGB, HLS, HSV, L*a*b, YCrCb, and (CIE)XYZ are the color representations that we used to test and evaluate fruit ripeness. Each color representation brought different results and we selected the best results or representations for each fruit.

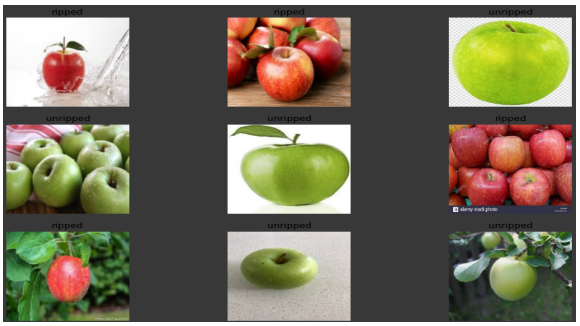


Figure 14 - RGB representation

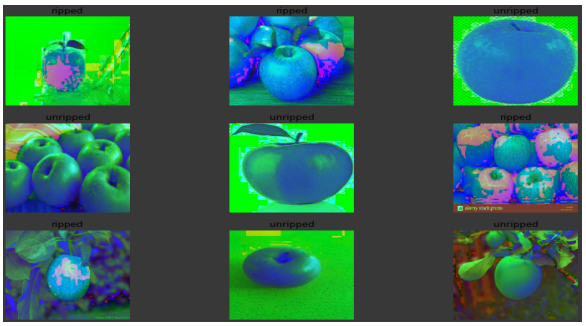


Figure 15 - HLS representation

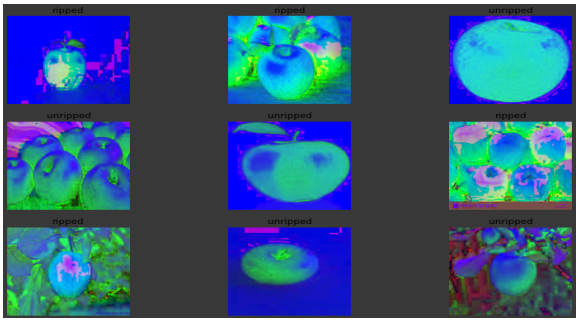


Figure 16 - HSV representation

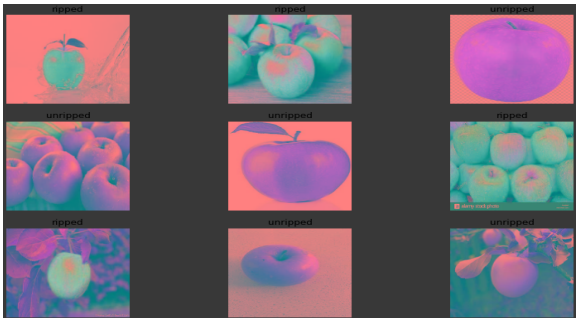


*Figure 17 - L*a*b representation*

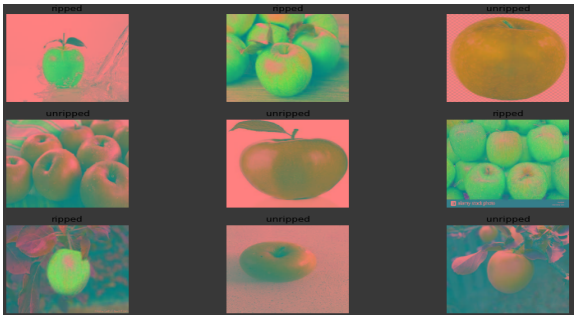


Figure 18 - YCrCb representation

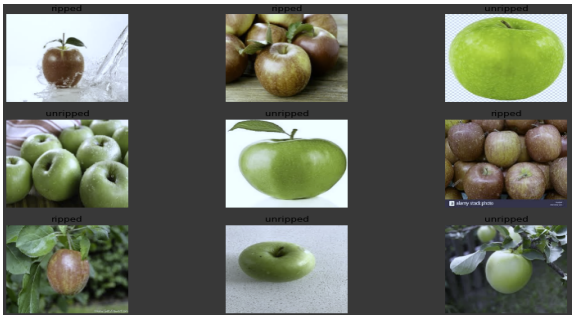


Figure 19 - XYZ representation

2.2.3 Server

While the object detection model will happen in real-time, we decided to deploy our fruit ripeness model to a server to relieve processing load from the mobile phone. We are using Flask to deploy our model to the server.

Flask is a web framework that is used to develop web applications using Python. In our case, we changed the usage so that Flask could be used to start a server using our Python code. In this file, we receive inputs from an HTTP request, load our pre-saved ripeness model, and return an output.

The process of uploading our model to a server first starts by saving our model in the SavedModel format. A SavedModel contains the TensorFlow Model as a .pb (protobuf) file, variables folder, and assets folder [44]. Once we have the saved model, we can easily load the model when it is needed in the future.

In order to start the server, we need to first create a virtual environment. Inside this environment, we can now install Flask and other necessary dependencies to run the Python file. Then, we can create a Flask instance that will initiate the start of the server. Figure 20 shows the complete flow of how the server is accessed. For detailed instructions on deploying the model, please take a look at *Appendix 6.8*.

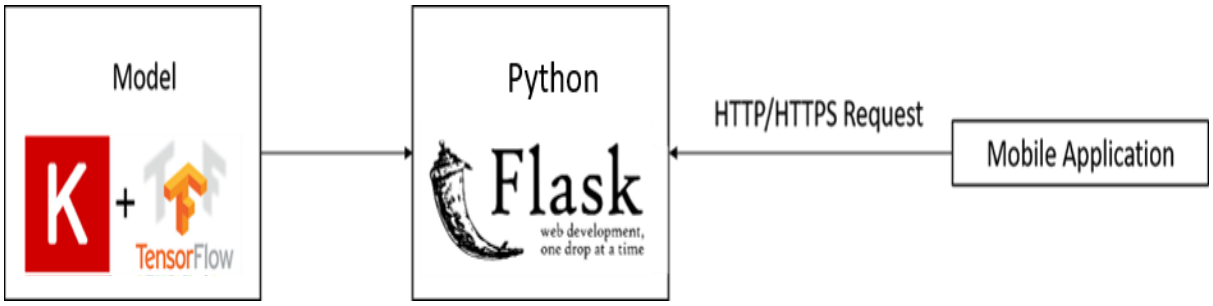


Figure 20 - Flask Diagram[45]

2.2.4 Application model

Hosting and running models natively in an application is a rigorous task. In order to make sure that the mobile application can work with models smoothly and efficiently, our team decided to develop a native application rather than a hybrid application. Native applications are built specifically for a particular operating system (IOS or Android), while hybrid applications are web applications packaged to be used as mobile apps. Although hybrid applications have the advantage of supporting both IOS and Android devices with one set of code, native applications run more smoothly and provide a better user experience [46]. Due to the heavy load of developing a native application, we decided to develop only for Android devices using Android Studio.

2.2.4.1 React Native

During the early stages of our project, our team first began the application development process on a platform called React Native. The main advantage of React Native is that it supports hybrid applications. This means it can be used to support both IOS and Android applications. However, our models were not able to run efficiently on React Native. React Native has a library for TensorFlow and for some TensorFlow Lite models. However, the model that our team utilized, YOLOv4, was not supported in React Native's TensorFlow Lite libraries. Even though we were able to run our YOLOv4 model through React Native's TensorFlow library, the greatest frame rate we achieved was 3 frames per second, which is barely usable. On the other hand, after moving to Android Studio and using it's TensorFlow Lite library, we were able to achieve frame rates of up to 50 frames per second. Therefore, we decided to switch to Android Studio.

2.2.4.2 Android Studio (Java)

Integration of custom models on Android device
YOLO

The first main component of the mobile application is successfully integrating our custom TensorFlow-Lite model into the application. The developers at TensorFlow-Lite already provide a template for doing this integration, but it is optimized to support integration for Mobilenet SSD models. In order to use our YOLO-based models, a new YOLOv4 class needs to be constructed. This is because there are two main parts that need to be included specifically for YOLO-based models: processing the output and performing non-maximum suppression. In terms of YOLO-based models, the output is returned as two separate 4-dimensional arrays. Therefore, we need to process these arrays separately and combine them together later.

The processing for these two arrays is the same. The initial shape of the arrays are [1, i, i , 45] where i = 13 for the first array and i = 26 for the second array. The arrays need to be then reshaped to [i, i , 3, 15]. The last dimension of size 15 contains the important information for our bounding boxes. The first two elements of the last dimension are the x and y of the

bounding boxes. The next two elements are the width and height of the bounding box. The fifth element is the objectness score which is the probability that an object exists in the bounding box. Finally, the last ten elements contain the probability that the fruit in the bounding box is the fruit in the specific classes we have predetermined.

A sigmoid function needs to be used as an activation function to extract the correct center coordinates and probabilities of the objectness score and classes [47]. Now we can finally set a threshold to remove all bounding boxes with low objectness scores and map out the rectangle for each bounding box. However, there may be multiple boxes that are referring to the same object. To solve this problem, non-maximum suppression is required. Non-maximum suppression is a way of getting the highest confidence bounding boxes and getting rid of any overlapping bounding boxes that are essentially targeting the same object. After doing non-maximum suppression, all that is left is to draw the bounding boxes on the camera preview.

Mobile-SSD

As described in Appendix 6.5, the Mobile-SSD model was first converted to the .tflite model to make the model compatible with the mobile application. There are two TensorFlow libraries that are needed to interpret the converted Mobile-SSD .tflite model, namely TensorFlow-Lite Interpreter and TensorFlow-Lite MetadataExtractor.

Interpreter library is needed for TensorFlow-Lite inference, where inference refers to the on-device execution of a TensorFlow-Lite model for making predictions based on input data[48]. To ensure minimum load, initialization, and execution latency, it employs static graph ordering and a custom memory allocator. There are four steps of using the Interpreter library: loading the.tflite model into memory, transforming the input data to make it consistent with the model, building interpreters and allocating tensors using the TensorFlow-Lite API, and interpreting the output.

In order for the Mobile-SSD model to work properly in the application, metadata was added to the .tflite model. Metadata provides a source of knowledge of what the model does and its input and output information. In other words, it provides a standard for model descriptions. The structure of the .tflite model with metadata is depicted in Figure 21 below.

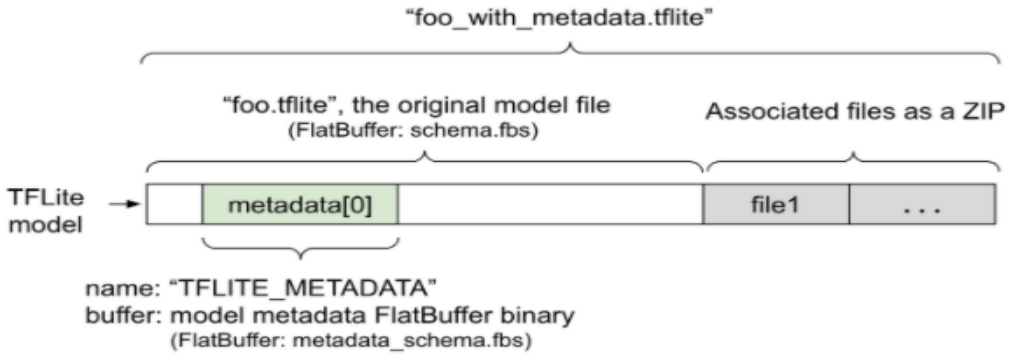


Figure 21 - Structure of tflite model with metadata[49]

The associated files are concatenated at the end of the .tflite file as a ZIP file. The associated files are the files that are related to the model, but in our case, we appended a classification label file which contains information about the fruit labels. The Interpreter library can consume this file format in the same way as before.

Since the SSD model was trained without quantization, we had to allocate the buffer for the input data with a size of (300 * 300 * 3 * 4), where 300 refers to the input size, 3 refers to the batch size, and 4 refers to the size of the float values. After interpreting the model with the generated input data, the output gave an accurate number of detected fruits, and for each detected fruit, it gave the top, left, right, bottom coordinates, a confidence level, and a fruit label, respectively.

2.2.4.3 Android Server - VOLLEY

Volley is a third party library recommended by Android Studio to handle HTTP Requests. Using the library is simple, but one thing to note is that our project requires us to send a large array for the model input. In order to do this, the cropped fruit image that will be processed for ripeness needs to be converted into a JSON object before being sent over to the server. Since the ripeness model requires an input array of size (1, 150, 150, 3), the cropped image bitmap that will be sent to the server is converted to RGB values in an array of size (1, 150, 150, 3). In order to save this array as a JSON object, each element of the RGB array needs to be mapped to a specific place in the JSON array. Since JSON objects are written as key/value pairs, each element in the RGB array can be assigned an incrementing key from 1 - 67500. When this JSON object is sent over the server to the Python file, the RGB array can be reconstructed by inputting each element in the JSON object one by one.

2.2.4.4 Application Flow and UI

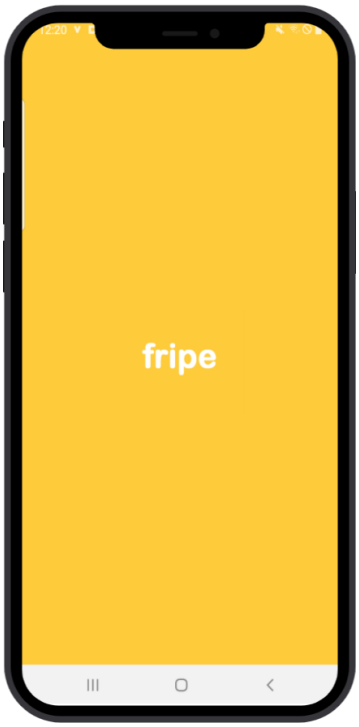


Figure 22 - Splash Screen

Splash Screen

The first screen that is displayed when the users open the application is this splash screen. Our application uses three main theme colors: orange, white, and black. These colors make a good combination to comfort the users. The splash screen also contains orange and white color to make it simple and advanced. It is also used to load and launch the contents of the application before showing them to the users.

Landing(Home) Page

This page is shown after all the contents are ready to be launched. The landing page consists of a bottom navigation view, a 'Real Time' button and a 'Gallery' button. The bottom navigation view has three components, namely Home, My Fruit, and Info. Each component leads to its respective pages upon pressing them. When the users click on the 'Real Time' button, the camera will be activated to detect fruits in real-time. Similarly when the users click on the 'Gallery' button, a gallery will be launched to detect fruits in the selected photo.

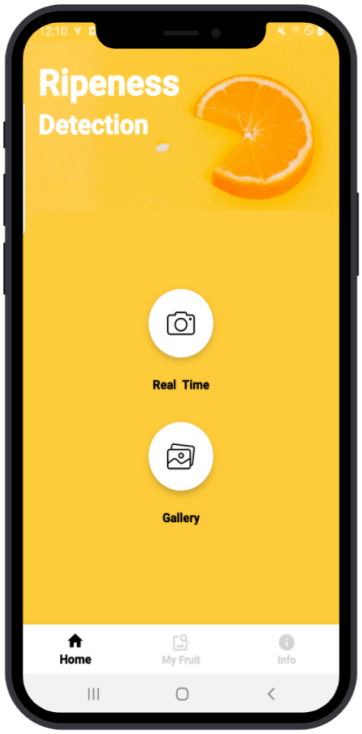


Figure 23 - Landing Page

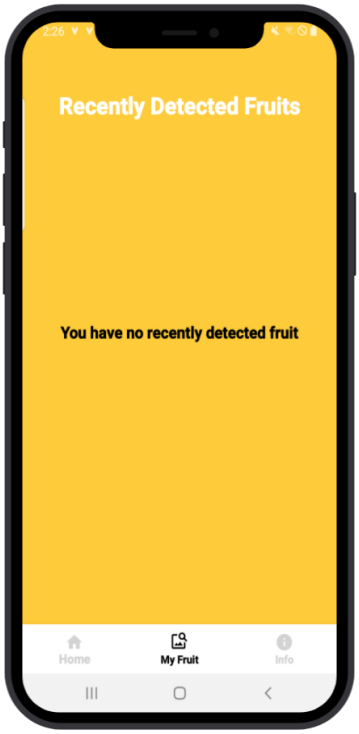


Figure 24 - 'My Fruit' Page

My Fruit Page

This page is launched when the users press on 'My Fruit' component of the bottom navigation view. My Fruit page shows images of recently detected fruits. If there is no recently detected fruit, the following text, "You have no recently detected fruit" is displayed.

Info Page

This page is launched when the users press on the Info component of the bottom navigation view. Info page displays a simple explanation about the application, as well as some tips while using the application.

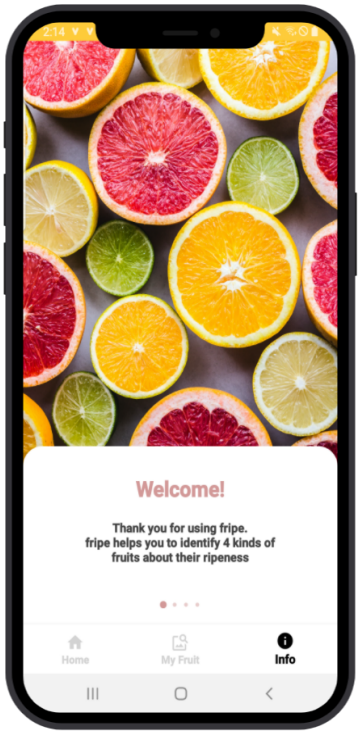


Figure 25 - 'My Fruit' Page

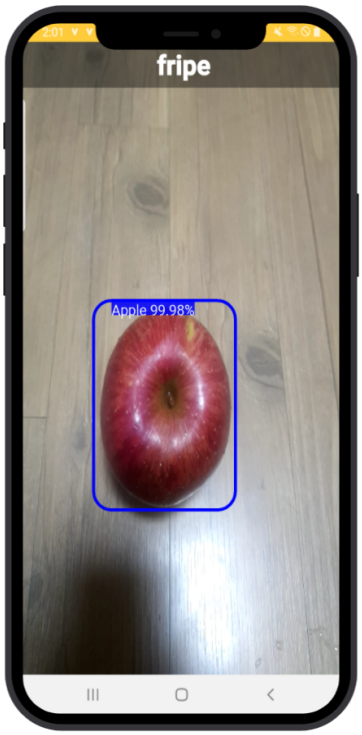


Figure 26 - Real time detection

Real Time Detection

This page is launched when the users click on the 'Real Time' button from the landing/home page. As soon as the camera is launched, the application automatically detects the fruits in the screen. To measure a specific fruit that the users want to know about its ripeness, they should click on the detected box to proceed to ripeness calculation.

Confirmation Dialog

To ensure that the detected type of fruit matches with the real type of fruit, the application asks the users whether it has correctly detected the fruit type. If the users click on the 'O' icon, it will proceed to calculate the ripeness of the selected fruit. If the users click on the 'X' icon, the application will ask the users to choose the correct type of fruit. The users can also close the dialog to select other fruit by touching the outside region of the dialog.

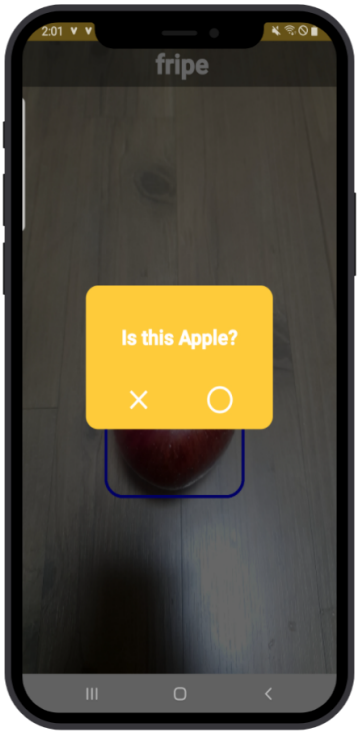


Figure 27 - Confirmation dialog

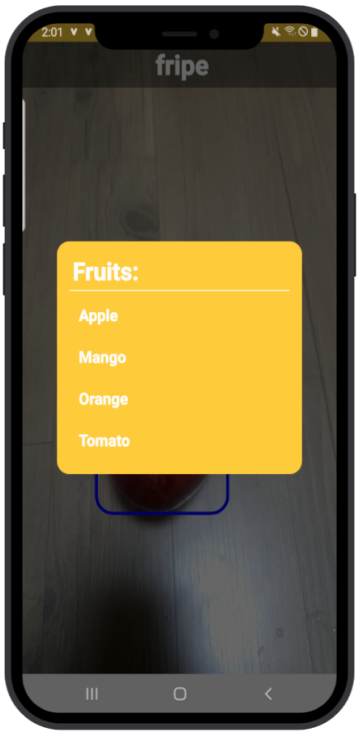


Figure 28 - Select fruit dialog

Select Fruit Type Dialog

As mentioned above, this dialog is shown when the user clicks on 'X' icon in the Confirmation dialog. If the users click on the correct fruit type, the application will proceed to ripeness calculation of the selected fruit.

Loading Page

Calculating ripeness takes a few seconds as the image has to be sent to the server. While the deployed model calculates the ripeness, the loading page is displayed to let the users know that the calculation is in process.

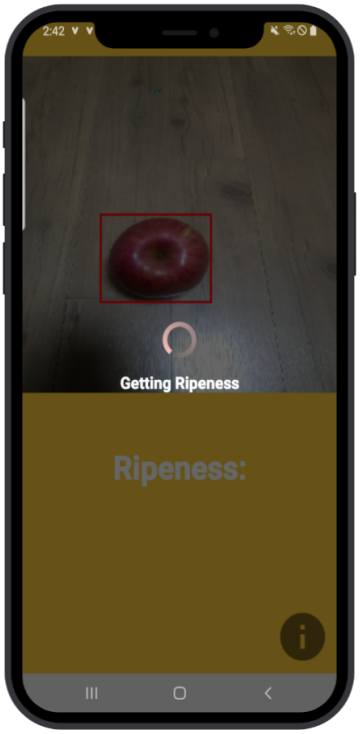


Figure 29 - Loading page

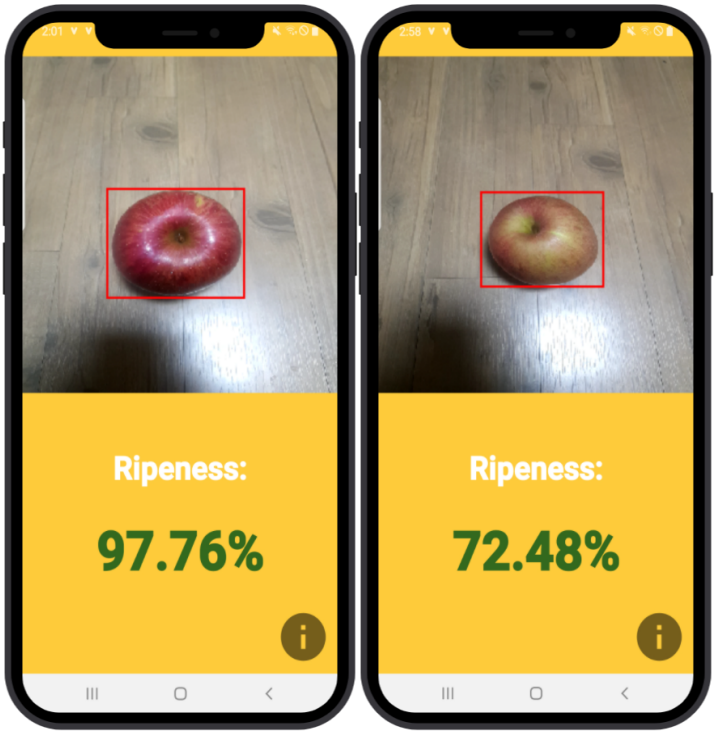


Figure 30 - Ripeness page

Ripeness Page

After the deployed model finishes calculating ripeness, the application successfully receives the output from the model, and the Ripeness page is shown to the users. The captured image is shown at the top of the page, and the percentage of ripeness is shown to the users upto 2~3 decimal places.

Attention Dialog

When the user clicks on the 'i' icon in the Ripeness page, an attention dialog is displayed to the users. This dialog contains the possible harmful effects of consuming unripe detected fruit, and numerous benefits of eating the detected fruit. The Attention dialog dismisses if the users clicks on the 'i' icon again.

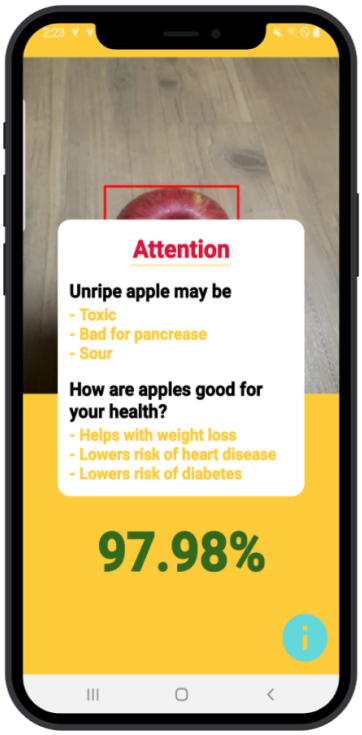


Figure 31 - Attention dialog

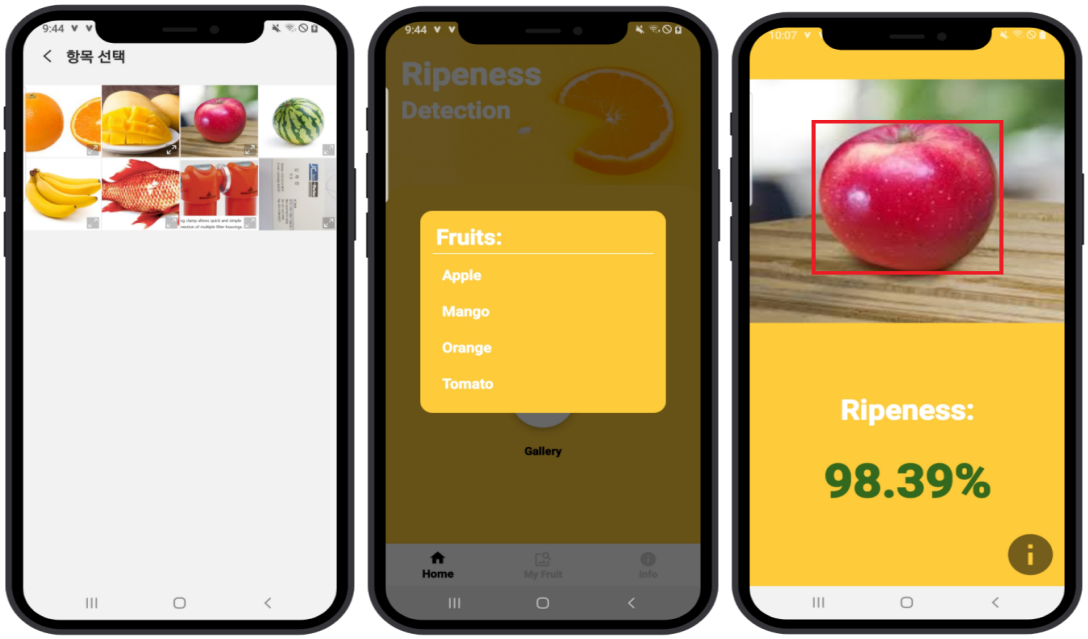


Figure 32 - Get ripeness from an image from gallery

If the users click on the 'Gallery' button from the landing page, they can choose one of the images from the gallery. When an image is chosen, our object detection model will detect the fruits in the image, and bounding boxes will be drawn around each fruit. From there, they can select a specific bounding box to get the ripeness of the fruit inside that bounding box. From then, the application works the same for calculating ripeness of the selected fruit in the image.

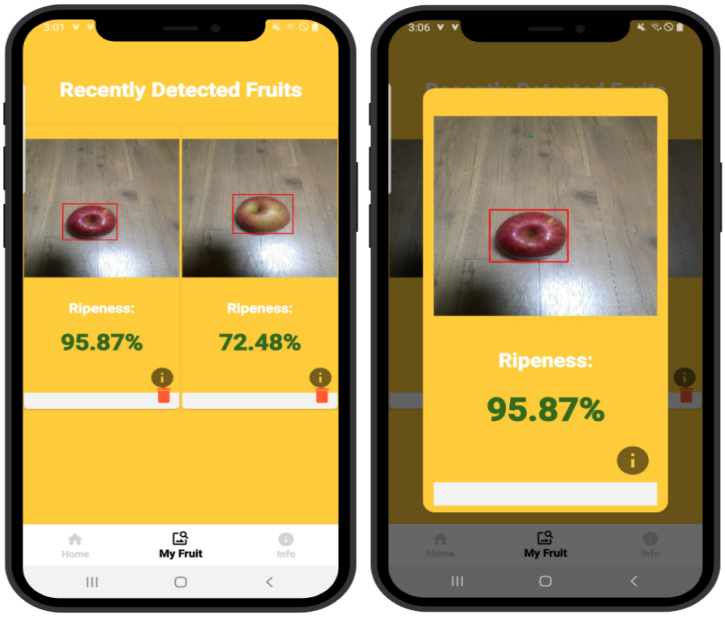


Figure 32 - 'My Fruit' page

After the users detect ripeness of the fruits, the detected images are automatically stored in the My Fruit page. Users can enlarge the stored image by clicking on the image, or they can delete the image by clicking on the trash icon at the right bottom of the image.

2.4.4.5 Mobile Application Evaluation

After changing our development platform from React Native to Android Studio, the performance of our real-time object detection increased by around 1500%. Although the main contribution of this increase was the efficiency of the model, it was also made possible because of the advantages that Android Studio has over React Native. The frame rate that our team was aiming for the real-time object detection was at least 10 frames per second. Since our real-time object detection has achieved a frame rate of around 25~50 frames per second, our real-time object detection has been more than successful.

Another section that is dependent on performance is calling the ripeness classification model from the server. Our team decided that a time of under 5 seconds needs to be accomplished in order to provide users the best experience. The server currently takes around 3~5 seconds to respond, so we have also fulfilled our goal in this area.

In the design perspective, we displayed our application to third-party groups in order to receive feedback. Although there are some areas of the application that can be refined, the overall consensus was that the design was satisfactory. Design is an area that continuously changes and improves, so we believe that our initial design is a good start.

2.3 Testing

Testing is a very important process that must be carried out to avoid uncertainties and systematic errors. By testing, recognizing, and fixing errors, we can validate our design and statements, making the result more reliable. The testing was done according to two main components, namely machine learning algorithms and application.

2.3.1 Ripeness classification model

Fruit Ripeness does not exactly correspond to the sugar content of fruits; however, in most cases, the sugar level increases as fruits ripen. Increasement of sweetness is a part of their ripening process [50].

Therefore, to test whether our ripeness models are accurate, we have prepared a glucose tester that can detect the sugar content of fruits. The specification of the device is shown below (Figure 33).

RHB-32 Specification

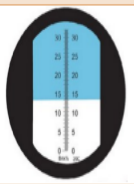
Model	RHB-32 ATC Refractometer
Measurement ranges	0-32% Brix
Accuracy	0.2% Brix
Resolution	±0.20% Brix
Scale view	
ATC	Yes
ATC temp.	10-30°C (50°F~86°F)
Material	Pure aluminum
Refractometer dimensions	170x32x32mm/6.7x1.3x1.3in
Refractometer net weight	128g/4.5oz
Package size	205x8x5.5cm / 8 * 3.5 * 2.3in
Package weight	282g/10oz

Figure 33 - specifications of RHB-32 device[51]

We conducted several tests with fruits that we have chosen and here are the results from ripeness classification models and the device.

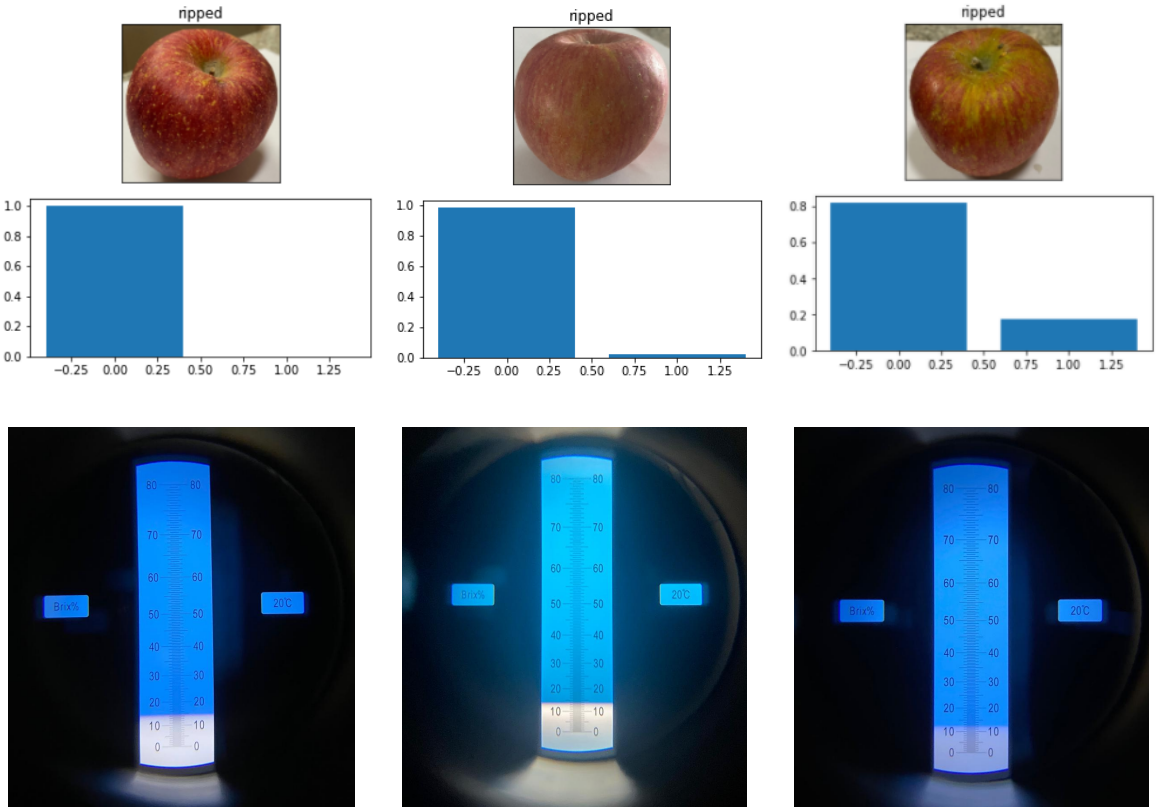


Figure 34 - Apple ripeness test

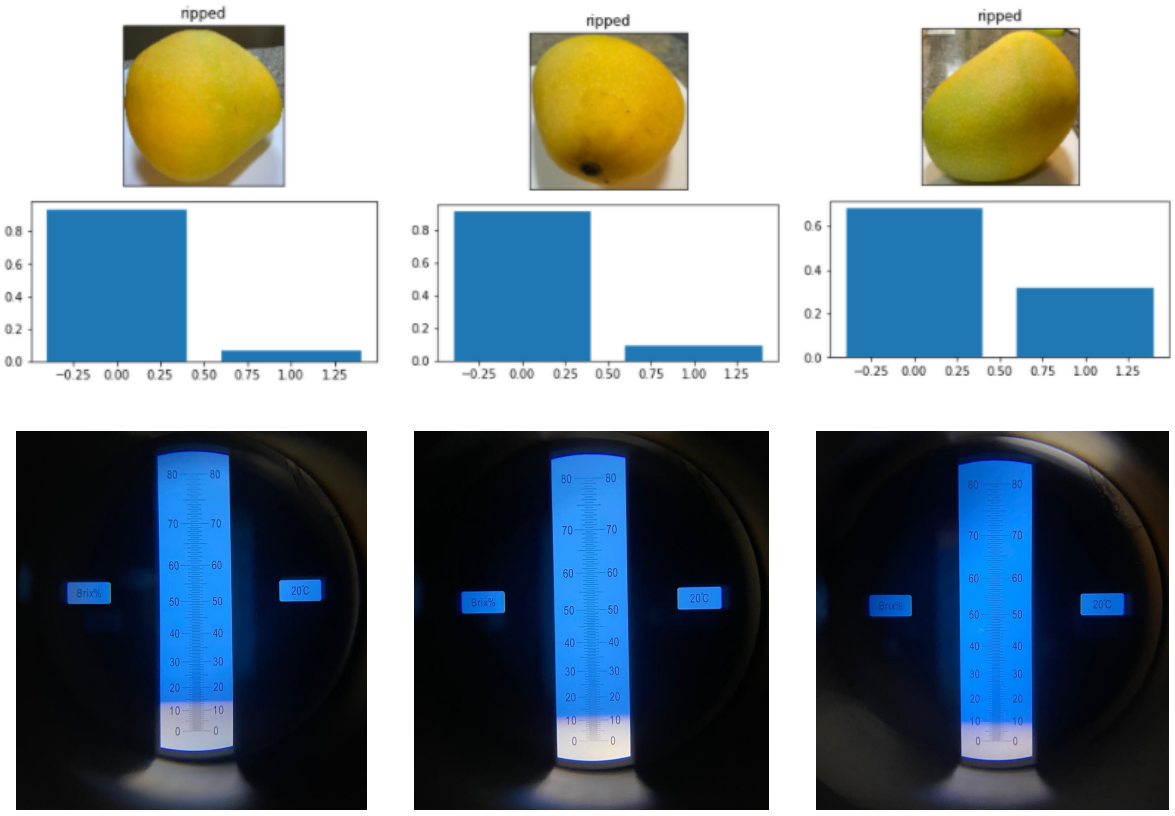


Figure 35 -Mango ripeness test

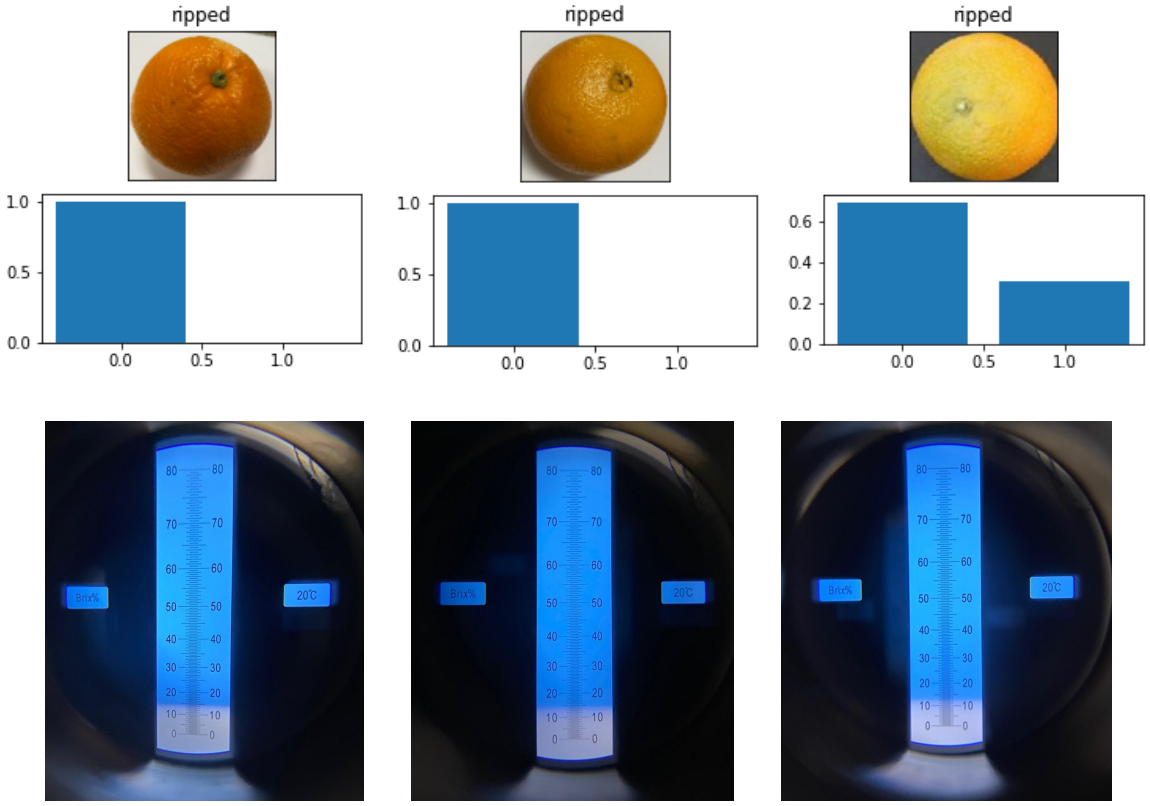


Figure 36 - Orange ripeness test

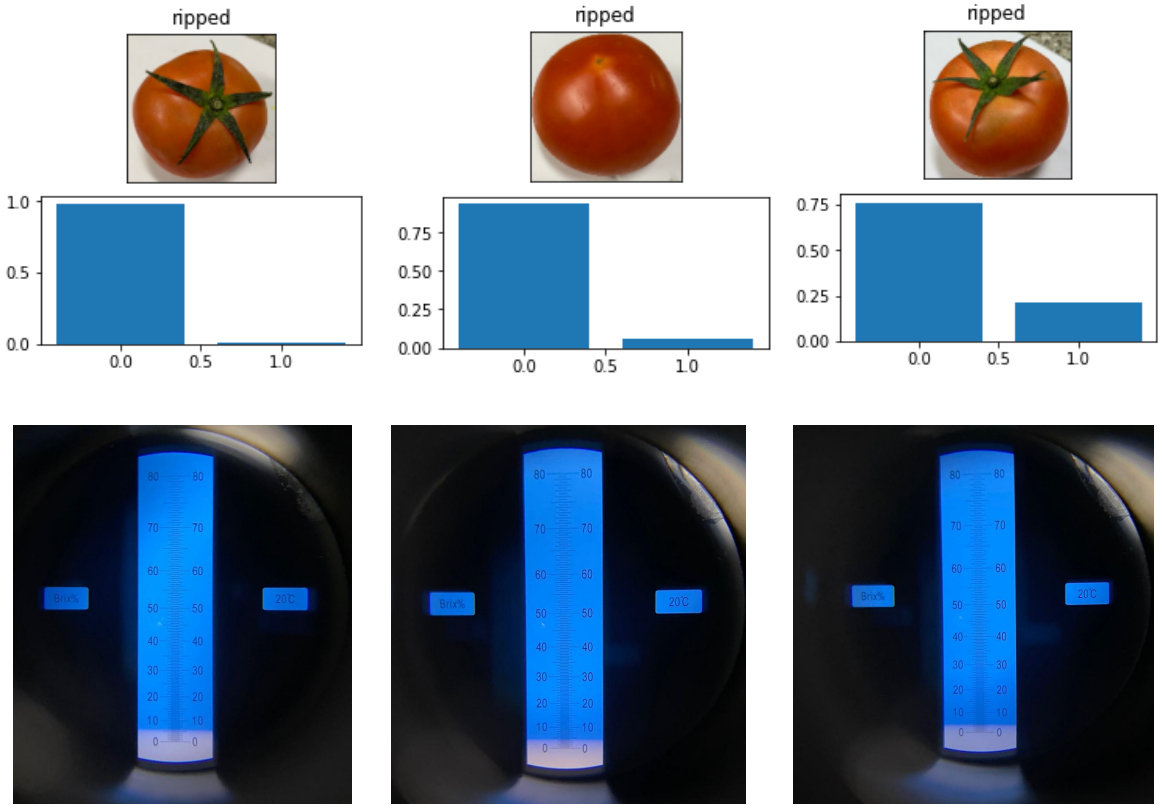


Figure 37 -Tomato ripeness test

From the above results (Figure 34-37), we are able to conclude that our ripeness classification models are pretty accurate in a sense that the fruits have higher Brix percentage when they are predicted as more ripped through the models.

2.3.2 Mobile application

There are several test cases to analyze the design and functionalities of the application. There should be a smooth conversion between the screens. Since the application has five to six different screens, there should be no lagging and delaying with this process. Similarly, there should also be a smooth user interaction with the features of the application. For example, when the users tap on the button to take a photo, there should be no discomfort in the experience of the users.

One of the most crucial parts of building the application is handling crashes. Proper error handling in development is necessary. We will test if there exists any crashes, especially when interacting with the server. Furthermore, the efficiency of each part of the application is important to test. We have already tested the frame rate for the real-time object detection functionality and the time it takes to get back results from the server in our application. Please refer to ***2.2.4.1 Android Studio*** for the detailed results of the tests.

The user interface is also an important feature in application development. The main UI component that we will focus on is the output. When the fruits are detected in real-time, the surrounding boxes are displayed just outside the fruits with the percentage of the ripeness at the top of the boxes. We will test whether the boxes and the ripeness are in the proper places with high stability. Furthermore, we plan to test the detection multiple times to make sure that the name of the detected fruits and the corresponding ripeness are constant over time.

2.4 Evaluation

Objective 1: Accurately identify fruits in real-time

By conducting feasibility tests for various object detection models, we have decided our model to be YOLO-fastest. The YOLO-fastest model used in this project was able to identify various fruits at an adequate speed that can support real-time detection on mobile devices. The model has achieved a mAP of 95% with a speed of 24 FPS on Galaxy s10 (snapdragon 855).

Objective 2: Precisely calculate the percentage of ripeness of fruits

To measure the ripeness of the fruits, we successfully have built separate ripeness models for each type of fruit. We implemented CNN as our main deep neural network. It allows to extract important features of each fruit to calculate the ripeness.

Objective 3: Provide a user-friendly and intuitive interface with high user-experience

The design was implemented after researching other similar applications and well-designed applications. Our team applied numerous design features and interfaces similar to those applications on top of our own advanced features and functions. We tested the application with some relatives and friends of our team to receive feedback and continuously improved to give the positive impacts on the users. At the end, we were able to implement a really simple user interface with a professional design that is comparable to other commercialized applications. We believe that our application can be used by a variety of people, including laymen, children, and elderly people.

2.5 Limitations and Further Study

Although we have accomplished all the objectives that we proposed, the application has some limitations and parts to improve on.

Tracking implementation

During the implementation for returning the ripeness score, we have discarded the initial application design of using tracking and chose a simpler solution to freeze. However, freezing the image is not efficient since it only allows the user to examine the ripeness of a single fruit, where with tracking, multiple fruits can be analyzed. Thus, in the future, it is preferable to implement tracking algorithms.

Tracking had two problems. First, the movement of the fruit object instance was hard to follow, which resulted in the tracker to fail. Second, it requires too much computational power resulting in application to lag while tracking.

To solve these issues, a faster object detection model is needed. By doing more research, we found that by quantizing the model using TFlite, it can improve the speed by 3 fold[52]. Thus, the first issue can be easily solved. Also, this allows us to use a simpler tracking algorithm. The simple online tracking algorithm (SORT), tracking algorithm we tested, consists of two simple algorithms: Kalman filter and Hungarian method. Kalman filter is an algorithm to estimate the position of the object in the future by using the position and velocity information. With the predicted position by Kalman filter and the actual position, Hungarian algorithm is used to associate instances. However, if the detection rate increases, we only have to use the Hungarian algorithms and compare the bounding box position of the previous frame and the current frame,to track objects. Since this reduces the computational power consumed by the Kalman filter, it will solve the lagging issue.

Accuracy of the ripeness model

One of the hardest parts of our project was to collect the image data, especially for the fruit ripeness model. Since the application has to detect whether the fruit is ripe or unripe, we had to have lots of unripe images of the fruits. However, it was extremely hard to get the unripe images as most of the fruits that we can find around us (supermarkets, fruit markets, etc) are ripped. Therefore, we strongly believe that our ripeness models could have much higher accuracies if greater amounts of unripe images were provided.

One solution is to visit fruit farms. We are able to see the whole ripening process of fruits in the fruit farms; therefore we can collect numerous unripe and ripped images. Another approach can be visiting grocery stores where sellers display all the sugar contents of fruits. Fruits with low glucose rates can be used as unripe data and high rates as ripped data. If we have time and chance to improve our application, we will use these solutions to further increase the accuracy of the ripeness models.

Types of fruits

When we go to a fruit market, we can find various types of fruits. However, our application can only process four types: apple, mango, orange, and tomato. We initially proposed to cover ten fruits, but implementing each type of fruit took much longer than what we expected. As a result, the convenience of the application has decreased.

If we had more time to carry out the project, we would add more types of fruits. We can simply implement the same terminologies to create object detection models and ripeness models for other types of fruits.

Mobile Application and Server

The current limitation of our mobile application system is that we currently only support Android devices. The downside of using a development platform for native applications is that if we wanted to make a similar application for IOS devices, we would have to repeat the same amount of work.

Another limitation is using Flask as our server. Although Flask allows for ease of use and can easily add more types of models into one python file, the disadvantage is that it needs to load the model everytime a request is called. This can decrease efficiency and increase the time it takes to get back ripeness data.

3. Conclusion

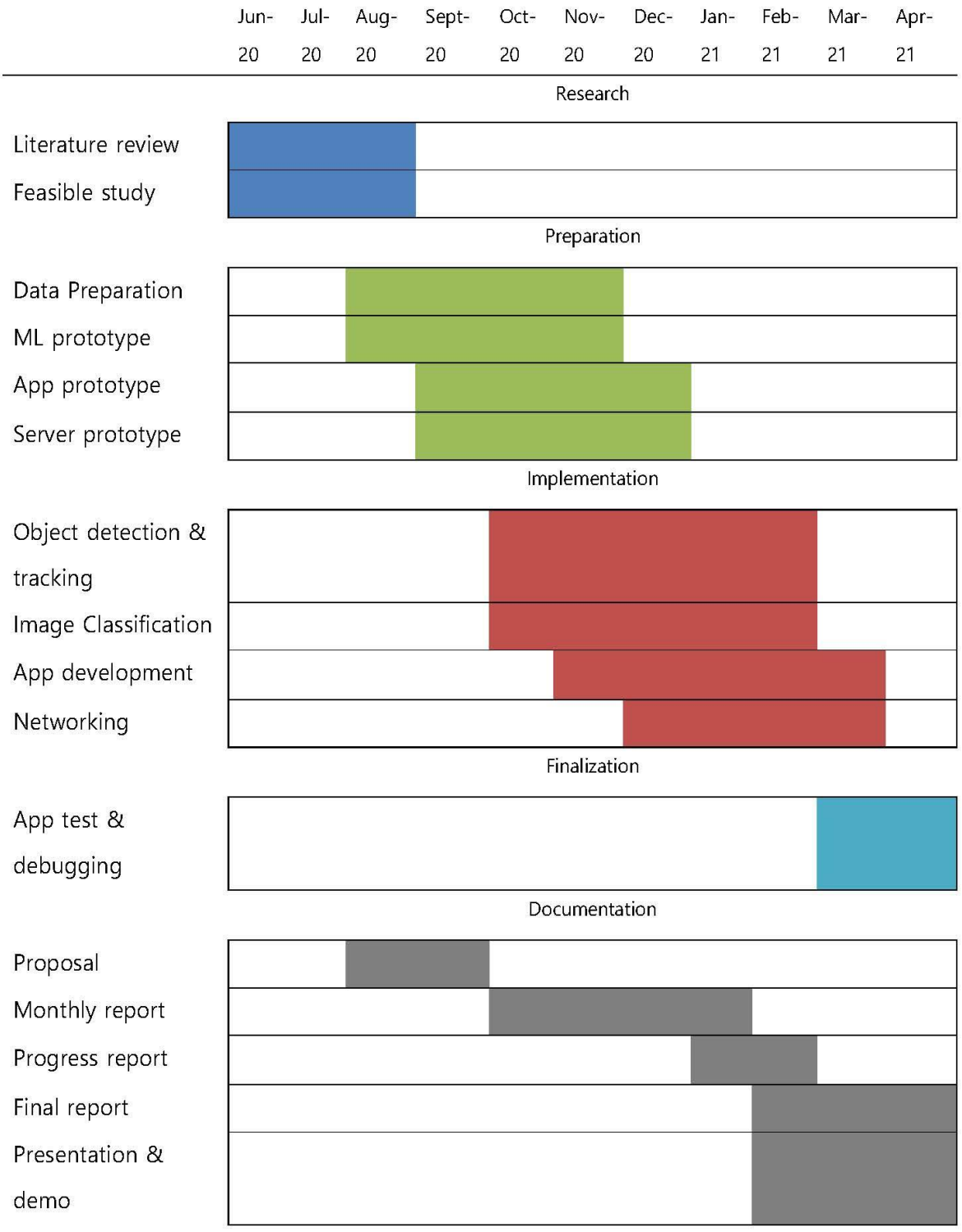
Our project has successfully implemented a complex integration between machine learning algorithms and a mobile device. Such a system allows users, especially households, to distinguish between ripe and unripe fruits in just a few seconds. Since the application will automatically detect the type of fruit within the camera frame, the system is simply to use for the user. The system also allows the user to either detect fruits in real-time or upload an image of a fruit that they have saved on their device. This increases the flexibility of the system and gives users the freedom to choose different routes to find a ripeness of a fruit. Providing the best environment for users extensively increases the chance for users to utilize the app to consume more fresh, sweet, and delicious fruits.

However, the current version of the project does come with some limitations. This includes the limited number of fruits that are currently available for detection and the variabel accuracy of the ripeness classification model due to lack of datasets. But the project has enormous potential to expand as the number of fruits can be increased by accumulating more data and can even be used to identify other food groups such as vegetables.

By implementing this project, we hope to grow awareness on eating healthy fruits and hope to impact society by allowing users to find fresh fruits in a more accurate and simple way.

4. Project Planning

4.1 Gantt Chart



4.2 Division of Work

T: Together L: Leader A: Assistant

	Taek Jung Kim	Junyoung Yoon	Jihoo Kim	Jingee Kim
Research				
Literature review	T	T	T	T
Feasible study	T	T	T	T
Preparation				
Data Preparation	L	L	A	A
ML prototype	L	L	A	A
App prototype	A	A	L	L
Server prototype	A	A	L	L
Implementation				
Object detection & tracking	L	A	A	A
Image Classification	A	L	A	A
App development	A	A	L	A
Networking	A	A	A	L
Finalization				
App test&debugging	A	A	L	L
Documentation				
Proposal	A	A	A	L
Monthly report	A	A	L	A
Progress report	A	L	A	A
Final report	L	A	A	A
Presentation & demo	T	T	T	T

5. Hardware & Software

Hardware				
Product	**Specification**	**Usage**	**price (HKD)**	**Quantity**
Mobile device	Specification may defer	Test application	700~10,000HKD	1
Google Cloud/ HKUST server	GPU server	Setup a server to process the data	Free	1
Laptop	ASUS Laptop	Train machine learning	Free	1
saccharimeter	Detecting sweetness of fruits	Testing ripeness model	120HKD	1

Software			
Product	**Version**	**Usage**	**Price**
TensorFlow	2.3	Machine learning	Free
CUDA, cudnn	10.2	Nvidia GPU architecture	Free
Darknet	-	Object detection	Free
OpenCV	4.5	Computer vision and machine learning software library	Free
CVAT	3.3	Online image labeling tool	Free
LabelImg	1.8	Image labeling tool	Free
Android Studio	4.0.1	Mobile Application	Free
React Native	0.63	Mobile Application Development	Free
TensorFlow Serving	2.4	Model Deployment Server	Free
Docker	20.10.2	Model Containerization	Free

6. Appendix

6.1 Image crawling[53]

```
1  !pip install selenium
2  import urllib.request #import urllib.request
3  from bs4 import BeautifulSoup #import BeautifulSoup
4  from selenium import webdriver #import webdriver
5  from selenium.webdriver.common.keys import Keys #import Keys
6  import time #import time; enablues you to use .sleep() while crwaling imges
7  # instantiate browser
8  binary = r'C:\yolov4\nextk\chromedriver.exe' #path to the chrome driver
9  browser = webdriver.Chrome(binary) #init browser
10
11 #get browser. this will open a new ready-for-searching tab of Chrome browser
12 browser.get("https://www.google.com/imghp?hl=en&search?hl=en&q=")
13
14 elem = browser.find_element_by_name("q") #init elem
15 elem.send_keys("unriped apple") #keywords that you wanna use to search
16 time.sleep(3)
17 elem.submit() #elem.submit()
18
19 # for-loop
20 for i in range(1 ,6):
21     # find body tag and execute send_keys(Keys.END) for i < 10 so 9 times
22     # Keys.END is when the END key is executed to be cliecked
23     browser.find_element_by_xpath("//body").send_keys(Keys.END)  #smb == when clicking show more result button
24     try:
25         browser.find_element_by_id("smb").click()
26         time.sleep(5)
27     except:
28         time.sleep(5)
29 time.sleep(5)
30
31 html = browser.page_source # get page_source from browser
32
33 # init soup by calling the method BeautifulSoup with the parameters; the variable html and "html.parser"
34 soup = BeautifulSoup(html, "html.parser") #this code enables you to fetch the image urls and download the images
35
36 # method for listing url
37 def fetch_list_url():
38     params = [] #declare and init an array
39
40     # find all img tags and the class whose name is rg_i
41     imgList = soup.find_all("img", class_ ="rg_i")
42
43     #Now, extract the img sources(urls) from imgList
44     for im in imgList:
45         try:
46             params.append(im["src"]) #source address of an img in the class rg_i
47         except KeyError:
48             params.append(im["data-src"])
49     return params #return params`
50
51 # method for downloading imgs from the url
52 def fetch_detail_url():
53     params = fetch_list_url() #init params by calling the method fetch_list_url()
54
55     # print(params)
56     a = 1 #a = 1    for p in params:
57         # @param p. the source img urls in params are assigned into p with this foreach-loop if urls are fetched properly
58         # @param path; gives download path
59         # @param a; gives auto-incrementing numeric file name
60         # finally, set .jpg extension to each of the img downloaded.
61         urllib.request.urlretrieve(p, r"C:\yolov4\image\ "+ str(a) + ".jpg" )
62         a+=1 #a = a + 1; increment nums
63
64 #by calling the method fetch_detail_url(), the method fetch_list_url() is executed first in it.
65 fetch_detail_url() #fetch_detail_url()
66 browser.quit() #close the browser
```

The image crawling code allows downloading images on the internet. By using the chrome driver, it opens a chrome web browser to search the keyword (line 15) on Google image. All the images shown on the google image page will be downloaded on a set directory (line 61).

6.2 Annotation format

```
<annotation>
  <folder></folder>
  <filename>apple.jpg</filename>
  <source>
    <database>Unknown</database>
    <annotation>Unknown</annotation>
    <image>Unknown</image>
  </source>
  <size>
    <width>1024</width>
    <height>768</height>
    <depth></depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>Apple</name>
    <occluded>0</occluded>
    <bndbox>
      <xmin>120.86</xmin>
      <ymin>229.9</ymin>
      <xmax>411.23</xmax>
      <ymax>535.67</ymax>
    </bndbox>
  </object>
  <object>
    <name>Apple</name>
    <occluded>0</occluded>
    <bndbox>
      <xmin>364.44</xmin>
      <ymin>68.27</ymin>
      <xmax>669.0</xmax>
      <ymax>375.2</ymax>
    </bndbox>
  </object>
  <object>
    <name>Apple</name>
    <occluded>0</occluded>
    <bndbox>
      <xmin>381.04</xmin>
      <ymin>352.71</ymin>
      <xmax>672.6</xmax>
      <ymax>641.9</ymax>
    </bndbox>
  </object>
  <object>
    <name>Apple</name>
    <occluded>0</occluded>
    <bndbox>
      <xmin>639.41</xmin>
      <ymin>201.01</ymin>
      <xmax>929.78</xmax>
      <ymax>502.0</ymax>
    </bndbox>
  </object>
</annotation>
```

```
0 0.259810 0.498418 0.283564 0.398138
0 0.504609 0.288717 0.297422 0.399648
0 0.514473 0.647533 0.284727 0.376549
0 0.766206 0.457689 0.283564 0.391914
```

Figure 38 - VOC format (left) and YOLO format (right) of image annotation label

We have used two different formats for labeling. VOC format, which is shown on the left part of Figure 38, is in an xml file that contains various information about the image. It contains image file name, dimension of the image, and bounding box information of the image (class name, x and y coordinates of top left and bottom right). YOLO format is a text file that each line represents an object instance. Each line has class number, x and y coordinate of the center of the box, width and height of the box (in percentage).

Example:

0.269810 = ((120.86 + 411.23) / 2) / 1024

(YOLO x) = (((VOC xmin) + (VOC xmax)) / 2) / (image width)

The code for conversion from VOC format to YOLO format is given by AlexyAB darknet on github repository[54].

6.3 Data augmentation for object detection dataset

There are two data augmentation methods we have used for training the object detection model, cutmix and mosaic.

CutMix is an augmentation method that crops segments from one image and pastes it onto the other image.

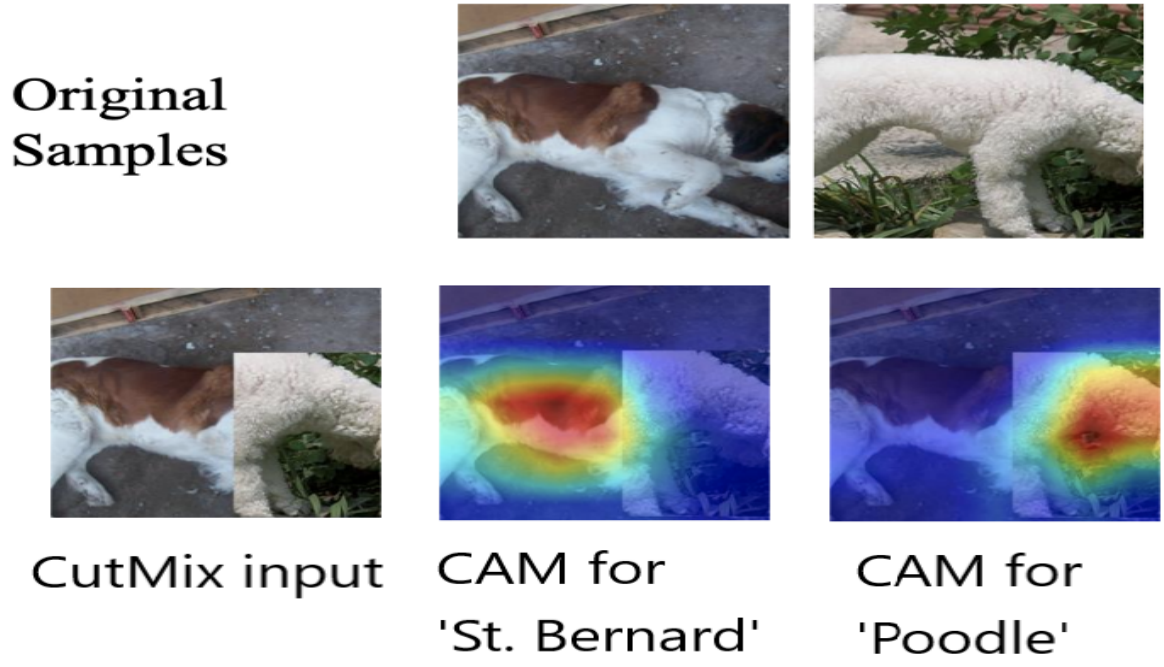


Figure 39 - Class activation mapping (CAM) for CutMix augmentation[55]

From Figure 39, the input image is a combination of St. Bernad's body with Poodle's head. By looking at the CAM, the model had correctly detected the body as a St. Bernard and head as a Poodle. By using this augmentation, it allows the model to learn various parts of the object[55].

Mosaic data augmentation combines 4 images into one. Since the combined image size stays the same, the object in the image becomes smaller. With mosaic augmentation, the model can learn to detect smaller objects improving performance[56].

6.4 Darknet installation guide

The installation guide is for a Windows 10 computer with CUDA supported Nvidia GPU. Below is a step by step installation procedure for Darknet[57-58].

1. Install Python version 3.7 or above
   a. Numpy module needed (in command prompt type *pip install numpy*)
2. Install Git, Cmake and Visual Studio 19
   a. For installation of Visual Studio, Check "Python development" and "Desktop development with C++"
3. Update GPU drivers
4. Install CUDA version 10.0 or above (note CUDA 10.1 is not compatible for Darknet)
5. Download CUDnn 7.0 or above (Have to match CUDA version you have installed)
   a. Copy contents from CUDnn (lib, bin and include folder) and paste it onto CUDA folder (directory path for CUDA is "C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2")
6. Install OpenCV version above 2.4 and OpenCV-contrib, create new folder name "build"
7. Cmake configuration of OpenCV
   a. Open Cmake, put OpenCV directory as source code, build binaries to be build directory made on step 8 (press configure)
   b. Generator must be Visual Studio 19 and optional platform for generator must be x64 (press finish)
   c. put OpenCV-contrib modules directory on OPENCV_EXTRA_MODULES_PATH (C:\opencv\opencv_contrib-master\modules) and press configure
   d. Check BUILD_opencv_world, WITH_CUDA, CUDA_FAST_MATH and press generate
8. Open ALL_BUILD.vcxproj on build folder
   a. Go to Build -> Batch Build, Check Debug and Release for ALL_BUILD and Install, Config must be x64
9. Open command prompt and Clone darknet repository *git clone https://github.com/AlexeyAB/darknet*
   a. copy OpenCV\build\bin\Release\opencv_videoio_ffmpeg450_64.dll, OpenCV\build\bin\Release\opencv_world450.dll, OpenCV\build\bin\Debug\opencv_world450d.dll, C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v10.2\bin\cudnn64_8.dll and pastest onto darknet\build\darknet\x64
   b. In darknet\build\darknet open darknet.vcxproj with notepad
      i. Change CUDA version to your version
   c. In darknet\build\darknet open darknet.sln, go to Project property,
      i. On C/C++ -> general ->additional include directory put openCV\build\install\include
      ii. On C/C++ ->preprocessor->preprocessor definition remove CUDNN_HALF
      iii. On CUDA C/C++->device remove compute_75 and sm_75
      iv. On linker->general->Additional Library directory add C:\opencv\build\install\x64\vc16\lib
      v. build darknet

6.5 SSD

6.5.1 xml to csv conversion

```python
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET

def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df


def main():
  image_path = os.path.join(os.getcwd(), 'images/train')
  xml_df = xml_to_csv(image_path)
  xml_df.to_csv('data/train.csv', index=None)

  image_path = os.path.join(os.getcwd(), 'images/test')
  xml_df = xml_to_csv(image_path)
  xml_df.to_csv('data/eval.csv',index=None)

main()
```

6.5.2 Generate tfrecord file

The Python code for generating tfrecord file from csv file, images, and labelmap file are given by tensorflow developers in tensorflow github repository[59].

6.5.3 transfer learning on pre-trained ssd mobilenet with generated tfrecord files

The Python code for the implementation of transfer learning on pre-trained ssd mobilenet with tfrecord files are in the official tensorflow github repository[60].

6.5.4 convert checkpoint file to protobuf file

The Python code for conversion of checkpoint file to protobuf file is given in the official tensorflow github repository[61].

6.6 Model conversion

Darknet to TensorFlow

To convert darknet YOLO model to tensorflow, github repository keras-YOLOv3-model-set[62] was used. This allows a darknet model (cfg and weights file) to be converted to a tensorflow understandable h5 file. By using h5, we were able to convert it to TensorFlow-Lite.

However, there are some changes that need to be made on the code. The code states that if the cfg file does not give the correct width and height, the input is set None. Also, the batch size is set to None as a default. However, TFlite model on android studio does not support dynamic-size-tensor causing the application to crash.

To solve this issue, in "keras-YOLOv3-model-set/tools/model_converter/convert.py"

```
115     print('Creating Keras model.')
116     if width and height and args.fixed_input_shape:
117         input_layer = Input(shape=(height, width, 3), name='image_input')
118     else:
119         input_layer = Input(shape=(None, None, 3), name='image_input')
120     prev_layer = input_layer
121     all_layers = []
```

Change line 115

```
115     print('Creating Keras model.')
116     input_layer = Input(shape=(416, 416, 3), batch_size=1, name='image_input')
117     prev_layer = input_layer
118     all_layers = []
```

By giving a set input size and batch size, the tensor has static size.

.pb to TFlite

The Python code for converting protobuf files to TensorFlow Lite file was imported from this github repository[63].

.h5 to TFlite

```
1  import tensorflow as tf
2  model = tf.keras.models.load_model('yolo-fastest.h5')
3  converter = tf.lite.TFLiteConverter.from_keras_model(model)
4  tflite_model = converter.convert()
5  # Save the model.
6  with open('model.tflite', 'wb') as f:
7      f.write(tflite model)
```

Conversion to Tensorflow.js

Conversion to Tensorflow.js follows the same process as above. First load the model. Then instead of using converter to convert to tflite, convert to tfjs.

```
import tensorflowjs as tfjs
tfjs.converters.save_keras_model(model, './weights')
```

6.7 Data cropping for ripeness classification model

The cropping augmentation has been done using a fruit detection model. By feeding fruit images to the fruit detection model, it will provide a bounding box for fruit instances present in the image. Since the bounding box is the smallest box that fits the fruit instance, cropping the image by the bounding box will produce data without any unwanted attributes.

```
!pip install opencv-python
import cv2
import os
import numpy as np
def getOutputsNames(net):
    layersNames = net.getLayerNames()
    return [layersNames[i[0] -1] for i in net.getUnconnectedOutLayers()]

def postprocess(img, outs, classIds, boxes):
    confidences = []
    for i in range(len(outs)):
        data = outs[i]
        increment = 0
        for j in range(len(outs[i])):
            increment += len(outs[i][j])
            scores = outs[i][j][5:]
            _, confidence, _, classIdPoint = cv2.minMaxLoc(scores)
            if (confidence > CONFTHRESHOLD):
                centerX = int(data[j][0] * len(img[0]))
                centerY = int(data[j][1] * len(img))
                width = int(data[j][2] * len(img[0]))
                height = int(data[j][3] * len(img))
                left = int(centerX - width/2)
                top = int(centerY - height/2)

                classIds.append(classIdPoint[0])
                confidences.append(confidence)
                boxes.append([left, top, width, height])
    indices = cv2.dnn.NMSBoxes(boxes, confidences, CONFTHRESHOLD, NMSTHRESHOLD)
    return indices

CONFTHRESHOLD = 0.5
NMSTHRESHOLD = 0.4
INPWIDTH = 416
INPHEIGHT = 416
classes = []
f = open("obj.txt", "r")
while (1):
    line = f.readline()
    if not line:
        break;
    classes.append(line[:-1])

directory = "original"
IMGLIST = []
for root,dirs,files in os.walk(directory):
    for file in files:
        IMGLIST.append(os.path.join(root,file))

for i in range(len(classes)):
    try:
        os.mkdir(os.path.join("crop",classes[i]))
    except OSError as error:
        print(error)

net = cv2.dnn.readNet("yolo-fastest_best.weights", "yolo-fastest.cfg")
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA_FP16)

x = 0
for i in range(len(IMGLIST)):
    img = cv2.imread(IMGLIST[i])
    blob = cv2.dnn.blobFromImage(img, 1/255.0, (INPWIDTH, INPHEIGHT), (0,0,0), True,
    False)
    net.setInput(blob)

    outs = net.forward(getOutputsNames(net))

    boxes = []
    classIds = []
    indices = postprocess(img, outs, classIds, boxes)
```

```
71
72          for j in range(len(indices)):
73              x += 1
74              top = max(boxes[np.squeeze(indices[j])][1],0)
75              bottom =
                min(boxes[np.squeeze(indices[j])][1]+boxes[np.squeeze(indices[j])][3],
                len(img))
76              left = max(boxes[np.squeeze(indices[j])][0],0)
77              right =
                min(boxes[np.squeeze(indices[j])][0]+boxes[np.squeeze(indices[j])][2],
                len(img[0]))
78              crop = img[top:bottom, left:right]
79              cv2.imwrite("crop/"+ classes[classIds[np.squeeze(indices[j])]] +
                "/"+str(x)+".jpg", crop)
80
81      while cv2.waitKey(10) < 1:
82          (grabbed, frame) = vc.read()
83          if not grabbed:
84              exit()
85
86          start = time.time()
87          classes, scores, boxes = model.detect(frame, CONFIDENCE THRESHOLD, NMS THRESHOLD)
88          end = time.time()
89
90          start_drawing = time.time()
91          for (classid, score, box) in zip(classes, scores, boxes):
92              color = COLORS[int(classid) % len(COLORS)]
93              label = "%s : %f" % (class names[classid[0]], score)
94              cv2.rectangle(frame, box, color, 2)
95              cv2.putText(frame, label, (box[0], box[1] - 10), cv2.FONT_HERSHEY_SIMPLEX,
                0.5, color, 2)
96          end_drawing = time.time()
97
98          fps label = "FPS: %.2f (excluding drawing time of %.2fms)" % (1 / (end - start),
            (end_drawing - start_drawing) * 1000)
99          cv2.putText(frame, fps_label, (0, 25), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 2)
100         cv2.imshow("detections", frame)
```

By placing the image data directory, label, config and weights file in the same directory, the code will generate a cropped image and save it in its corresponding fruit directory.

6.8 Using Flask to Serve our Model

The steps to get started with Flask is documented clearly in the Flask website [64]. This section of the appendix will describe how to set up Flask in relation to our project.

Before using Flask, we first need to set up a virtual python environment and install Flask in this environment. This can be done by following the instructions written in the Flask website [64]. By setting up a virtual environment, we will be able to set the Python version and library versions to exactly what we need.

Our team has created a separate Python file that can receive inputs through HTTP requests, load the model, and produce an output. This file contains TensorFlow, Keras, and OpenCV libraries. Therefore, we also need to install these libraries into our virtual environment. In order to install all these libraries, make sure to have a version of Python that is 3.6 or greater.

Now we can include the following code to set up the Flask server:

```
app = Flask(__name__)
CORS(app)

@app.route("/", methods = ['POST', 'GET'])
def predict():
```

The first line is simply creating a Flask instance with the name of our web application. The second line allows requests to be made on the server without interference from security protocols. Now, the third line of code is telling Flask what URL should be used for the server. For example, since our web application is currently routed to "/", our URL will be http://domainname/. If our web application was routed to "/example", the URL would be http://domainname/example. Finally, we can include our code to run our model in the predict function, and call this function when the URL is called upon.

Now all we need to do is run our Python file, and the server will begin to start running.

```
(venv) C:\Users\jinge\Documents\flask>python model.py
2021-04-13 23:09:24.614923: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could
 not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2021-04-13 23:09:24.616303: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cud
art dlerror if you do not have a GPU set up on your machine.
 * Serving Flask app "model" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
2021-04-13 23:09:31.916098: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could
 not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2021-04-13 23:09:31.916528: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cud
art dlerror if you do not have a GPU set up on your machine.
 * Debugger is active!
 * Debugger PIN: 781-759-604
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

6.9 Meeting Minutes

6.9.1 1st meeting

Time: June 15 2020, 3~4pm Zoom meeting

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

Dr. Tsoi gave a brief introduction about FYP.

Informed us about the topic (predicting student's performance)

Discussed what kind of data is needed to predict student's performance

- Lists of courses and instructors (A range, UST space reviews etc)
- Relevancy of the courses (prerequisites, similarities)
- Student's past grade (from CSE department)
- Student's personality, interests, other aspects

Decided to get a student's grade from the CSE department (Dr. Tsoi will request) and conduct a survey to get other data.

6.9.2 2nd meeting

Time: June 30 2020, 8~9:30pm

Attending: Taek Jung, Junyoung, Jihoo, Jingee

Discussion:

The main problem that we had with determining a student's performance using machine learning was that getting data was extremely difficult. However, we got a message from Dr. Tsoi that his request (students' grade) to the CSE department was rejected. That means the most reliable data source for this project was gone.

1. Discussed if it is possible to obtain student's grades by conducting a survey, but we could not be sure that the surveyee will be honest about their grade.

2. Concluded that we might not be able to get the correct model for predicting student performance.

We decided to come up with several topics 1~2 per each member. The following were the results that we gathered:

- Ripeness of fruits
- Predicting sport team
- Youtube related video algorithm
- Informing when is right time to sleep (for insomnia)
- Matching bencher business?
- Voice recognition

6.9.3 3rd meeting

Time: July 6 2020 7:30~8:30pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

We presented our ideas to Dr. Tsoi to find the best one we should do for our FYP project. Although we had many ideas that we could do, we were worried about whether some projects may be too easy or too difficult. So, we asked Dr. Tsoi's opinions on the project ideas that we came up with.

Dr. Tsoi mentioned that some of the ideas might be too difficult for us to finish in a year. He recommended that we either do the "Informing the right time it is to sleep" project or the "Ripeness of fruit" project.

After our meeting with Dr. Tsoi, we continued to have a meeting between ourselves. We discussed the following:

1. After hearing Dr. Tsoi funneled our options to two projects, we discussed which project would be the best one to do.

2. We decided that fruit ripeness would be the most interesting and the easiest to gather data.

3. We mentioned that each person should do some research regarding machine learning and fruit ripeness.

6.9.4 4th meeting

Time: July 27 2020 8:00~9:00pm with ourselves

Attending: Taek Jung, Junyoung, Jihoo, Jingee

Discussion:

After doing research about our project, we decided to meet up because it was time to start writing our proposal.

1. We discussed the different parts each person will take up in writing the proposal.
    a. Jingee would write the overview, objectives, and literature review.
    b. Junyoung would write the methodology portion.
    c. Jihoo would write the design portion.
    d. Taek Jung would write methodology about collecting data and related algorithms. Also construct the Gantt Chart.
2. We also discussed potential methods and algorithms to use in our project.
    a. We talked about how we could use TensorFlow-Lite to integrate our trained model with our mobile application.
    b. We discovered that we could use YOLO (You Only Look Once) Algorithm to use in object detection.
3. Concluded the meeting by deciding that each person would start writing their part and also work on machine learning algorithms.

6.9.5 5th meeting

Time: August 17 2020 8:00~9:00pm with ourselves

Attending: Taek Jung, Junyoung, Jihoo, Jingee

Discussion:

We demonstrated our object detection prototype, SSD mobile net, to Dr, Tsoi. Even though we have implemented on mobile net, the inference time was extremely long which was not suitable on mobile devices. Dr. Tsoi suggested:

1. Use smaller object detection model that can work on mobile devices
    a. Use existing API
    b. Make new model
2. Use server for processing machine learning algorithms

Another problem encounter was inefficient to manually label the data. We have demonstrated usage of LabelImg application. Dr. Tsoi gave us a method, auto-annotation, that is used to label dataset in the machine learning field. First trains a model with small data and uses it to annotate the entire dataset.

6.9.6 6th meeting

Time: Sep 2 2020 7:30~8:30pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

We roughly finished the proposal and sent the document to Dr. Tsoi before the meeting day. The introduction and the content of the project were written in detail, but other details were not fully completed. The purpose of this meeting was to receive feedback and suggestions to help us complete and further develop the quality of the proposal.

We discussed various ways to implement the objectives of our project. We shared our ideas with Dr. Tsoi as well as among team members. The supervisor respected every idea from us, and he kindly told us the ways to further improve and enhance the idea. The main feedback and suggestions from the supervisor are the following:

1. Other than ML Kit as one of our tools to implement the project, the supervisor suggested using the server provided by HKUST CS System.

2. After discussing the design of the UI of the application, Dr. Tsoi provided feedback to use the software designing tool to implement the design.

3. Dr. Tsoi shared the key points of the previous proposals to refer to.

6.9.7 7th meeting

Time: Oct 20 2020 7:30~8:30pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

In this meeting, we discussed with our supervisor on moving our mobile application to React Native in order to accommodate IOS devices. Since Android Studio can only support Android devices, we wanted to move to React Native. Also, Dr. Tsoi also gave us some tips on writing our first monthly report. We also showed him our model working on our mobile application for the first time. The model loading and the actual prediction was relatively slow, so we also discussed ways to increase the efficiency such as:

1. Reducing the size of the model
2. Utilizing the GPU of the phone more

We also showed him our object detection model by using apples, and discussed how we could increase the accuracy of our model since the current accuracy was only 90%.

6.9.8 8th meeting

Time: Nov 4 20 2020 7:30~8:30pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

At this point of our project, we realised that a way to reduce the size of our model was using TensorFlow and TensorFlow-Lite. Furthermore, Junyoung had a prototype of his fruit ripeness model that could predict the ripeness of bananas. We showed the progress of both these models to Dr. Tsoi. Then, we discussed how we could upload Junyoung's model to the server. Dr. Tsoi recommended using HKUST's server, and provided an email address that we could contact to get information on using HKUST's server.

6.9.9 9th meeting

Time: Jan 6 2021 7:30~8:30pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

This was the first meeting we had after winter break. This meeting was about how much progress we should make by the progress report deadline. Dr. Tsoi recommended that we should finish around 70~80% of our project before the deadline. We then gave some updates on what we did over winter break such as model enhancements and real-time camera integration in our application.

6.9.10 10th meeting

Time: Jan 13 2021 7:30~8:30pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

This was a relatively short meeting, and it was a meeting to gain some feedback on Taek Jung's machine learning model. At first, Taek Jung showed how much progress he made on his model, and the current stumbling blocks he was facing. After hearing our progress, Dr. Tsoi suggested maybe including more functions that would be helpful to users, such as healthy recipes that show after displaying the ripeness of a fruit. He also gave very helpful information on ways to deploy our model on a server.

6.9.11 11th meeting

Time: Jan 28 2021 8:30~10:00pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

The purpose of this meeting was to give a general progress check to Dr. Tsoi. We showed him our model working in real-time and calling our model from a server in our application. Junyoung also showed Dr. Tsoi his model in action by using some ripe and unripe apples as input. Our supervisor then gave some advice and suggestions on what to consider when making our model better. One of his suggestions was not to use apple images that are obviously unripe and can be determined through the human eye. This defeats the purpose of using a machine-learning model to find ripeness in fruit.

Furthermore, Dr. Tsoi reminded us to start working on our progress report. He helped us set up a timeline in order to finish the progress report in time, and we decided to have our next meeting on February 8, 2021.

6.9.12 12th meeting

Time: Feb 8 2021 8:00~9:30pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

This was our meeting to discuss our progress report with our supervisor. Dr. Tsoi helped us understand what kind of content is necessary in the progress report. He also asked us to update him on our current progress.

After working through the whole winter break, we were able to show him some updates such as the work real-time object detection in our mobile application, and also using Docker to serve our model in the server.

We also discussed our current limitations and the inefficiency of our current model. Our maximum frame rate was 3 fps at this point, and we reassured Dr. Tsoi that we were looking for other ways to improve the efficiency of our model.

6.9.13 13th meeting

Time: Mar 9 2021 8:00~9:30pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

In this meeting, the first thing we discussed was the comments we received from our second reader. This meeting was not long after we received our grade for our progress report, and Dr. Tsoi congratulated us for our good work. He also mentioned that we should pay careful attention to the second reader's comments, because they were good comments that we needed to think about while developing our project.

We also talked about the limitations of gathering data for our project. Since we do not have access to large amounts of real fruits for data collection, our main source was the internet. But many of the sources from the internet are low quality and are not very diverse. We discussed with Dr. Tsoi about the possibility of reducing the number of fruits we would use in our application.

6.9.14 14th meeting

Time: Mar 17 2021 8:00~9:30pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

After our last meeting, most of our time spent was labeling new types of data. Because our collection of data was too few, we told Dr. Tsoi that all four of us focused our time on data labeling. While doing data labelling, we also realized the limitations of doing some types of fruits in our project, and discussed with Dr. Tsoi on the types of fruits we will now be focusing on.

6.9.15 15th meeting

Time: Mar 23 2021 8:00~9:30pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

Most of this meeting was about showing the prototype of our mobile application. We showed him our real-time object detection and Dr. Tsoi gave us some advice on how to improve the performance.

This meeting was also after our team decided to switch from React Native to Android Studio as our mobile application development platform. We explained to him the reasons we were changing and got his approval.

He also reminded us that since our final report is coming up soon, we should start finalizing the User Interface of our mobile application.

6.9.16 16th meeting

Time: Apr 6 2021 2:00~3:30pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Dr. Desmond Tsoi (supervisor)

Discussion:

We have shown our application with some of the features we have added. Dr. Tsoi liked the feature about user interaction (whether our application has correctly classified the fruit). He pointed out that we should put more time on designing UI/UX since some of the features looked crude. He recommended us to see some of the similar applications available on google playstore to get some ideas.

We discussed our final report. He recommended adding a table which compares our application and the existing method of determining ripeness. He said that this will help to justify why we developed this application.

6.9.17 17th meeting

Time: Apr 12 2021 7:15~8:15pm with supervisor

Attending: Taek Jung, Junyoung, Jihoo, Jingee, Dr. Desmond Tsoi (supervisor)

Discussion:

We updated our supervisor on the User Interface of our application and received some feedback. We also showed him the efficiency of our real-time fruit detection. Dr. Tsoi recommended that we change the number of times the bounding box of our fruit detection model is updated. The reason he gave for this is to provide a better user experience for users.

We also discussed our final report. As we finished writing our final report, our team asked some questions that we had while writing the final report.

7. References

[1] “Black Ice Software, LLC - Leader in Document Converter, Printing, Faxing and Imaging Technology,” *HSI Color Conversion - Imaging toolkit feature - Document Imaging SDK - Black Ice Software*. [Online]. Available: http://www.blackice.com/colorspaceHSI.htm. [Accessed: 14-Apr-2021].

[2] N. A. Zaidi, M. W. Tahir, M. J. Vellekoop, and W. Lang, “A Gas Chromatographic System for the Detection of Ethylene Gas Using Ambient Air as a Carrier Gas,” *MDPI*, 07-Oct-2017. [Online]. Available: https://www.mdpi.com/1424-8220/17/10/2283. [Accessed: 14-Apr-2021].

[3] Serena Mani Washington University in St. Louis, “Is It Safe to Eat Unripe Fruit? These Are the Fruits You Can and Can't Eat,” *Spoon University*, 22-Aug-2017. [Online]. Available: https://spoonuniversity.com/lifestyle/is-it-safe-to-eat-unripe-fruit-we-list-which-you-can-and-can-t-eat. [Accessed: 14-Apr-2021].

[4] “Identifying Color Differences Using L*a*b* or L*C*H* Coordinates,” *Konica Minolta Sensing*, 18-Mar-2021. [Online]. Available: https://sensing.konicaminolta.us/us/blog/identifying-color-differences-using-l-a-b-or-l-c-h-coordinates/. [Accessed: 14-Apr-2021].

[5] W. Castro, J. Oblitas, M. De-la-Torre, C. Cotrina, K. Bazán, and H. Avila-George, “Using machine learning techniques and different color spaces for the classification of Cape gooseberry (Physalis peruviana L.) fruits according to ripeness level,” 2019.

[6] AmazonWebServices, “Clarifruit: Helping to Develop a Global Quality Standard for the World's Fruits & Vegetables,” *YouTube*, 01-May-2019. [Online]. Available: https://www.youtube.com/watch?v=qfwycjG6LwM&t=31s. [Accessed: 14-Apr-2021].

[7] “Taking Photos,” *clarifruit*. [Online]. Available: https://www.support.clarifruit.com/takingphotos. [Accessed: 14-Apr-2021].

[8] “The technology for a quality standard for fruits and vegetables,” *Clarifruit*, 21-Mar-2021. [Online]. Available: https://www.clarifruit.com/solutions/technology/. [Accessed: 14-Apr-2021].

[9] “Clarifruit quality standard test for fruit & vegetables,” *Clarifruit*, 21-Mar-2021. [Online]. Available: https://www.clarifruit.com/solutions/product/. [Accessed: 14-Apr-2021].

[10] I. Fadelli, “FoodTracker: An AI-powered food detection mobile application,” *Tech Xplore - Technology and Engineering news*, 26-Sep-2019. [Online]. Available: https://techxplore.com/news/2019-09-foodtracker-ai-powered-food-mobile-application.html. [Accessed: 14-Apr-2021].

[11] Ipvideomarket, “Frame Rate Guide for Video Surveillance,” *IPVM*, 18-Jan-2021. [Online]. Available: https://ipvm.com/reports/frame-rate-surveillance-guide. [Accessed: 14-Apr-2021].

[12] “About,” *OpenCV*, 04-Nov-2020. [Online]. Available: https://opencv.org/about/. [Accessed: 14-Apr-2021].

[13] “Install (and Compile) OpenCV with Android Studio || Android Deep Learning with OpenCV #2,” *YouTube*, 07-May-2019. [Online]. Available: https://www.youtube.com/watch?v=7J4b0Djcips. [Accessed: 14-Apr-2021].

[14] “TensorFlow Lite on GPU,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/lite/performance/gpu_advanced. [Accessed: 14-Apr-2021].

[15] Tensorflow, “tensorflow/examples,” *GitHub*. [Online]. Available: https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/android. [Accessed: 14-Apr-2021].

[16] “Top 20 Fruits and Vegetables Sold in the U.S.,” *2019 | Produce Marketing Association*. [Online]. Available: http://www.pma.com/content/articles/top-20-fruits-and-vegetables-sold-in-the-us. [Accessed: 14-Apr-2021].

[17]M. Savchenko, “What Is the Most Popular Fruit in the World? Top 5 Fruits Ranked,” *Flo*. [Online]. Available: https://flo.health/health-articles/lifestyle/nutrition/what-is-the-most-popular-fruit-in-theworld. [Accessed: 14-Apr-2021].

[18] TasteAtlas, “50 Most Popular Fruits (types And Products) in The World,” *World Food Atlas: Discover 11,062 Local Dishes & Ingredients*, 13-Jan-2021. [Online]. Available: http://www.tasteatlas.com/50-most-popular-fruits-in-the-world. [Accessed: 14-Apr-2021].

[19] A. Mala, “Most Popular Fruits In The World,” *WorldAtlas*, 18-Aug-2020. [Online]. Available: http://www.worldatlas.com/articles/the-most-popular-fruit-in-the-world.html. [Accessed: 14-Apr-2021].

[20] A. Dubner, “The Most Delicious Fruits,” *Ranker*, 27-Feb-2020. [Online]. Available: http://www.ranker.com/list/most-delicious-fruits/analise.dubner. [Accessed: 14-Apr-2021].

[21] *Open Images Dataset V6*. [Online]. Available: https://storage.googleapis.com/openimages/web/visualizer/index.html?set=train&c=%2Fm%2F06mf6. [Accessed: 14-Apr-2021].

[22] M. Oltean, “Fruits 360,” *Kaggle*, 18-May-2020. [Online]. Available: http://www.kaggle.com/moltean/fruits. [Accessed: 14-Apr-2021].

[23] C. Gorgolewski, “Fruit Recognition,” *Kaggle*, 04-Feb-2020. [Online]. Available: http://www.kaggle.com/chrisfilo/fruit-recognition. [Accessed: 14-Apr-2021].

[24] bangkokandmore, “FRUIT MARKET in Bangkok Thailand | Amazing Tropical Fruits,” *YouTube*, 19-Apr-2020. [Online]. Available: http://www.youtube.com/watch?v=4OdvvHOGeHg. [Accessed: 14-Apr-2021].

[25] Snacktopia, “Fruitmarket In Hong Kong,” *YouTube*, 05-Mar-2015. [Online]. Available: http://www.youtube.com/watch?v=qjdBI31nlxM. [Accessed: 14-Apr-2021].

[26] PoonCfu 新翠工程公司, “Strongest fruit market at Hong Kong 最強水果市場 大成街街市,” *YouTube*, 31-Aug-2020. [Online]. Available: https://www.youtube.com/watch?v=RZffrag6dF8. [Accessed: 14-Apr-2021].

[27] NO잼형제쿠킹&먹방, “구월동 농산물시장 구경하기 과일 채소 엄청 싸요^^,” *YouTube*, 17-Nov-2018. [Online]. Available: http://www.youtube.com/watch?v=a9bXX8SEeDw. [Accessed: 14-Apr-2021].

[28] tokian100, “신세계백화점 본점 식품관 구경 VLOG,” *YouTube*, 27-Jun-2019. [Online]. Available: http://www.youtube.com/watch?v=jEe2o64NuIY. [Accessed: 14-Apr-2021].

[29] Openvinotoolkit, “openvinotoolkit/cvat,” *GitHub*. [Online]. Available: https://github.com/openvinotoolkit/cvat. [Accessed: 14-Apr-2021].

[30] Tzutalin, “tzutalin/labelimg.” [Online]. Available: https://github.com/tzutalin/labelImg. [Accessed: 14-Apr-2021].

[31] AlexeyAB, “AlexeyAB/darknet,” *GitHub*. [Online]. Available: https://github.com/AlexeyAB/darknet. [Accessed: 14-Apr-2021].

[32] Dog-Qiuqiu, “dog-qiuqiu/Yolo-Fastest,” *GitHub*. [Online]. Available: https://github.com/dog-qiuqiu/Yolo-Fastest. [Accessed: 14-Apr-2021].

[33] J. Hui, “SSD object detection: Single Shot MultiBox Detector for real-time processing,” *Medium*, 15-Dec-2020. [Online]. Available: https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06#:%7E:text=SSD%20is%20a%20single%2Dshot,offsets%20to%20default%20boundary%20boxes. [Accessed: 14-Apr-2021].

[34] “Google Colaboratory,” *Google*. [Online]. Available: https://colab.research.google.com/notebooks/intro.ipynb. [Accessed: 14-Apr-2021].

[35] “TFRecord and tf.train.Example ： TensorFlow Core,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/tutorials/load_data/tfrecord#:%7E:text=Download%20notebook,to%20understand%20a%20message%20type. [Accessed: 14-Apr-2021].

[36] Tensorflow, “tensorflow/models,” *GitHub*, 28-Jul-2020. [Online]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md. [Accessed: 14-Apr-2021].

[37] Tensorflow, “tensorflow/models,” *GitHub*. [Online]. Available: https://github.com/tensorflow/models/tree/master/research/object_detection/configs/tf2. [Accessed: 14-Apr-2021].

[38] “Training checkpoints : TensorFlow Core,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/guide/checkpoint#:%7E:text=Checkpoints%20capture%20the%20exact%20value,objects)%20used%20by%20a%20model.&text=The%20SavedModel%20format%20on%20the,the%20parameter%20values%20. [Accessed: 14-Apr-2021].

[39] S. Yohanandan, “mAP (mean Average Precision) might confuse you!,” *Medium*, 09-Jun-2020. [Online]. Available: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2#:%7E:text=The%20mean%20Average%20Precision%20or,an%20IoU%20threshold%20of%200.5. [Accessed: 14-Apr-2021].

[40] *F-measure and Fbeta-measure for the Machine Learning Model*. [Online]. Available: https://onlinehelp.explorance.com/blueml/Content/articles/getstarted/mlcalculations.htm?TocPath=Get+started%7C_____3. [Accessed: 14-Apr-2021].

[41] “(PDF) Flower species recognition system using convolution neural networks and transfer learning,” *ResearchGate*. [Online]. Available: http://www.researchgate.net/publication/320746968_Flower_species_recognition_system_using_convolution_neural_networks_and_transfer_learning#pf3. [Accessed: 14-Apr-2021].

[42] T. S. T. Srivastava, “ANN Algorithm: How Artificial Neural Network Works,” *Analytics Vidhya*, 26-Jul-2020. [Online]. Available: http://www.analyticsvidhya.com/blog/2014/10/ann-work-simplified. [Accessed: 14-Apr-2021].

[43] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way,” *Medium*, 17-Dec-2018. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed: 14-Apr-2021].

[44] “Using the SavedModel format : TensorFlow Core,” *TensorFlow*. [Online]. Available: http://www.TensorFlow.org/guide/saved_model. [Accessed: 14-Apr-2021].

[45] V. Kurama, “Deploying Deep Learning Models on the Web With Flask,” *Paperspace Blog*, 09-Apr-2021. [Online]. Available:

https://blog.paperspace.com/deploying-deep-learning-models-flask-web-python/. [Accessed: 14-Apr-2021].

[46] *Medium*. [Online]. Available: https://uxplanet.org/native-vs-hybrid-mobile-apps-heres-how-to-choose-192ecbf04da8. [Accessed: 14-Apr-2021].

[47] "How to Implement a YOLO (v3) Object Detector from Scratch in PyTorch: Part 1," *KDnuggets*. [Online]. Available: https://www.kdnuggets.com/2018/05/implement-yolo-v3-object-detector-pytorch-part-1.html. [Accessed: 14-Apr-2021].

[48] "TensorFlow Lite inference," *TensorFlow*. [Online]. Available: https://www.tensorflow.org/lite/guide/inference. [Accessed: 14-Apr-2021].

[49] "Adding metadata to TENSORFLOW LITE MODELS." [Online]. Available: https://www.tensorflow.org/lite/convert/metadata. [Accessed: 14-Apr-2021].

[50] S. S. Wiley, "Does Sugar Content Rise as Fruit Ripens?," *LIVESTRONG.COM*. [Online]. Available: https://www.livestrong.com/article/555468-does-sugar-content-rise-as-fruit-ripens/. [Accessed: 14-Apr-2021].

[51] YHEQUIPMENT CO., LIMITED, "RHB-32 Refractometer Datasheet," *Gaming Law Review and Economics*, vol. 20, no. 10, pp. 859–868, 2016.

[52] Tensorflow, "Post-training quantization : TensorFlow Lite," *TensorFlow*. [Online]. Available: http://www.tensorflow.org/lite/performance/post_training_quantization. [Accessed: 14-Apr-2021].

[53] 오쉬 사용자, "31. 웹크롤링[이미지]," *itopia*, 25-Mar-2019. [Online]. Available: https://osh88itopia.tistory.com/86. [Accessed: 14-Apr-2021].

[54] AlexeyAB, "AlexeyAB/darknet," *GitHub*. [Online]. Available: https://github.com/AlexeyAB/darknet/tree/master/build/darknet/x64/data/voc. [Accessed: 14-Apr-2021].

[55] L. H. Trung, "https://euroasia-science.ru/pdf-arxiv/the-controllability-function-of-polynomial-for-descriptor-systems-23-31/," *EurasianUnionScientists*, vol. 4, no. 65, 2019.

[56] J. Solawetz, "Data Augmentation in YOLOv4," *Medium*, 12-Oct-2020. [Online]. Available: https://towardsdatascience.com/data-augmentation-in-yolov4-c16bd22b2617. [Accessed: 14-Apr-2021].

[57] Augmented Startups, “YOLOv4 Tutorial #1 - Prerequisites for YOLOv4 Installation in 10 Steps,” *YouTube*, 13-May-2020. [Online]. Available: http://www.youtube.com/watch?v=5pYh1rFnNZs&t=764s. [Accessed: 14-Apr-2021].

[58] Augmented Startups, “YOLOv4 Tutorial #2 - How to Run YOLOv4 on Images and video,” *YouTube*, 18-May-2020. [Online]. Available: http://www.youtube.com/watch?v=sUxAVpzZ8hU. [Accessed: 14-Apr-2021].

[59] Tensorflow, “tensorflow/docs,” *GitHub*, 18-Sep-2020. [Online]. Available: https://github.com/tensorflow/docs/blob/master/site/en/tutorials/load_data/tfrecord.ipynb. [Accessed: 14-Apr-2021].

[60] Rensorflow, “legacy · tensorflow/tree/research · GitHub.” [Online]. Available: https://github.com/tensorflow/models/tree/master/research/object_detection/legacy. [Accessed: 14-Apr-2021].

[61] Tensorflow, “tensorflow/models,” *GitHub*. [Online]. Available: https://github.com/tensorflow/models/tree/master/research/object_detection. [Accessed: 14-Apr-2021].

[62] david8862, “david8862/keras-YOLOv3-model-set,” *GitHub*. [Online]. Available: https://github.com/david8862/keras-YOLOv3-model-set. [Accessed: 14-Apr-2021].

[63] andresubagjamanurung, “pb_to_tflite.py,” *Gist*. [Online]. Available: https://gist.github.com/andresubagjamanurung/1eea9662e3aa297678c84c0afb41ecc1. [Accessed: 14-Apr-2021].

[64] “Installation¶,” *Installation - Flask Documentation (1.1.x)*. [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/installation/#python-version. [Accessed: 13-Apr-2021].