FINAL PROJECT REPORT - COMP6065001

"Image Classification of Fresh and Rotten Fruits using Deep Learning"

Arvin Yuwono (2502009721)	arvin.yuwono@binus.ac.id
Christopher Owen (2502019180)	christopher.owen002@binus.ac.id
Justin Theofilus Yonathan (2502036382)	justin.yonathan@binus.ac.id
Pandya Limawan (2502022433)	pandya.limawan@binus.ac.id

BINUS INTERNATIONAL

FACULTY OF COMPUTING AND MEDIA

COMPUTER SCIENCE

2023

TABLE OF CONTENTS

PROBLEM DESCRIPTION

Humans are required to consume healthy foods to boost their activities throughout the day. In this modern era, people prefer to buy freshly made foods rather than raw ones because they often do not want to think about and prepare their own foods. This behavior makes it difficult for them if one day they need to make their own dish because it is rare for them to do grocery shopping. They will have a hard time distinguishing between different types of raw fruits and vegetables, as well as their edible condition, whether they are fresh or rotten. A great concern is that this would deter humans from purchasing and consuming fruits and vegetables, which are of utmost importance to the human body.

People need to learn to choose raw foods, such as fruits and vegetables, to become familiar with recognizing raw products. The problem is that it is hard to just have one perspective on them. People need to be aware of their appearance, aroma, and feel because there are some cases where the color and size look fine but are not very fresh. Smelling can also be a challenge because it is hard to notice if the person’s nose is not well trained for that kind of situation. Feeling the texture might not always help in determining which, because firm and soft can be both situations depending on the kind of fruit or vegetable.

To solve this issue, the team proposes a solution using artificial intelligence technologies to help humans specify whether raw foods are fresh or rotten. One of the methods of artificial intelligence is deep learning, which utilizes the CNN derivative algorithm with additional tools and libraries to further train the model using a large dataset.

SOLUTION FEATURES

Algorithms

- **MobileNetV2**

  MobileNetV2 is a pre-trained deep learning model developed by Google based on the Convolutional Neural Network (CNN) algorithm. Compared to other models, MobileNet, as its name suggests, was designed to even be able to run smoothly on mobile devices. Aside from its simplicity, the group decided to choose MobileNetV2 due to its natural advantage of not learning noises, allowing it to have better generalized performance. In addition, it is designed to have a lightweight and efficient architecture, making it suitable for resource-constrained devices, such as smartphones. Its architecture is optimized for performance while maintaining a good balance between accuracy and model size. It also has a low memory footprint compared to a normal large CNN, which can be deployed on limited RAM devices. Furthermore, the efficiency of MobileNetV2 allows for real-time inference on mobile devices, where quick and accurate predictions are desired for applications with image classifications (Ahmed, 2023).

  Within the project, MobileNetV2 is being used to recognize the fruits. Additionally, it adds and trains additional layers on top of this model to teach it how to classify different fruits based on the training images. After training, the model saves this newly trained model to use it for recognizing images of fruits efficiently.

Additional Technologies

- **(TensorFlow 2 and Keras)**

  The Google Brain team created the open-source TensorFlow machine learning framework. An eager execution mode for more intuitive development, a streamlined API, and improved support for dynamic computation graphs are just a few of the enhancements brought about by TensorFlow 2. It provides a comprehensive ecosystem of tools to perform robust experimentation and model deployment in production on any platform. Moreover, it simplifies the API by cleaning up deprecated APIs and reducing duplication, and it works efficiently with multi-dimensional arrays. It also supports easy model building with Keras integration to make it seamless and straightforward.

  Keras is a high-level interface for creating neural networks. It simplifies building, training, and deploying deep learning models within the TensorFlow framework. In addition, it always gets the best performance in models when running on a GPU. This integration combines Keras' user-friendly design with TensorFlow's powerful features for efficient and effective model development.

  In the project, Keras is utilized to leverage the MobileNetV2 architecture as a pre-trained model for image recognition. Keras serves as the interface to access and customize this pre-built neural network, allowing the addition of custom layers and enabling training on specific datasets for fruit image classification. It handles model compilation, training, and conversion to TensorFlow Lite format for deployment on mobile devices.

User Interface

- **FastAPI**

FastAPI is an advanced web framework in Python designed for creating APIs. It merges the simplicity of Flask with the speed of asynchronous programming, using Python's type hints to automatically produce comprehensive API documentation. Its asynchronous capabilities enable handling multiple requests concurrently, ensuring efficiency for demanding applications. Known for its ease of use, swiftness, and automatic validation of data, FastAPI streamlines API development, offering both speed and reliability (FastAPI, n.d.).

Python FastApi is used as the main module to run the web interface for our model's implementation. Within the FastApi code, two endpoints are defined, which are index (/) and predict (/predict). Inside the index page, an HTML template is rendered that asks the user to choose an image from their devices, allowing for camera usage on mobile phones as well. Afterwards, when the user inputs an image and clicks the "Predict" button, then the model will be loaded using the tf.lite.interpreter() method. Before processing the input image using the pretrained model, the input image is reformatted to suit the requirements of the model, converted into RGB, resized to 224 x 224 format, put in image array format, and normalized. Next, the input is processed by the model, and the output of the model is returned. Finally, this will lead to the "/predict" page, which will display the input fruit along with the confidence rate of the prediction for the input fruit.

- **Bootstrap**

Bootstrap stands as a freely accessible and open-source front-end development toolkit, serving as a robust framework facilitating the construction of websites and web applications. Its primary focus lies in fostering the creation of mobile-first, responsive web content by offering an extensive assortment of pre-designed templates. By furnishing developers with a comprehensive array of standardized syntax and components, Bootstrap streamlines the process of crafting visually appealing and adaptable interfaces. This toolkit's core essence revolves around empowering users to swiftly create websites that seamlessly adapt to various devices, ensuring a consistent and engaging user experience across multiple platforms.

For this project, the use of bootstrap can be seen inside the html template for both index (/) and predict (/predict) pages. A link to a bootstrap stylesheet URL is linked, which provides access to the bootstrap class styles. The reason for choosing Bootstrap as the styling tool of choice is due to its responsive properties and simplistic method. Hence, it allows a simpler procedure for styling the HTML template.

- **Render**

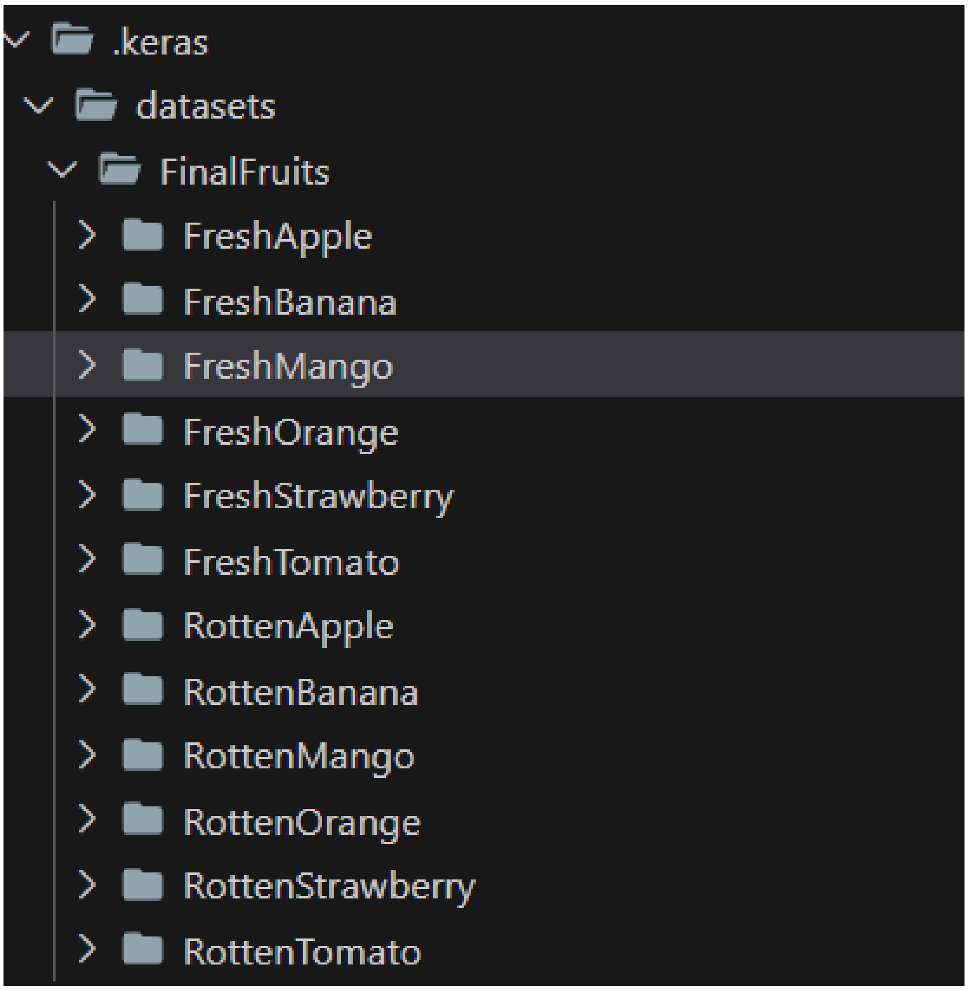
Render is a user-friendly cloud platform that simplifies hosting and scaling web applications and services. It streamlines deployment by allowing developers to deploy code from Git repositories and automatically scaling applications as needed.

With support for various languages and frameworks, Render minimizes infrastructure management, enabling developers to focus on building their projects.

Within the scope of the project, Render is used for the deployment of the web app interface. It includes free HTTPS, TLS, and domains. The features from Render have allowed the team to stay focused on building the project instead of using DevOps. The process of deploying to Render involves setting up the build configurations, such as the run command and build command to execute, as well as creating a txt file containing the dependencies required to run the code. Afterwards, render will start installing the specified dependencies and build the app on their cloud server.

SOLUTION DESIGN ARCHITECTURE

Data Preparation



```
.keras
  datasets
    FinalFruits
      FreshApple
      FreshBanana
      FreshMango
      FreshOrange
      FreshStrawberry
      FreshTomato
      RottenApple
      RottenBanana
      RottenMango
      RottenOrange
      RottenStrawberry
      RottenTomato
```

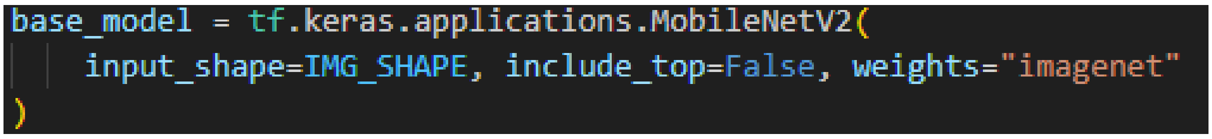
Several datasets are downloaded from Kaggle. The team extracted the zip file and combined the same fruits with the same condition (fresh or rotten) into one folder. To better understand the file structure, please refer to the image above. There are two types of fruits: fresh and rotten. The team also selected and managed six different fruits: apple, banana, mango, orange, strawberry, and tomato. In total, the dataset is in 12 folders with a total of more than 35,000 pictures. The training and validation sets are split by an 8:2 ratio. In addition, the team managed a separate test set for 50 pictures in each folder, for a total of 600 pictures.

Several such folders would then be compressed into ZIP. To access the dataset, we use pathlib.Path.home as the root directory, then combine it with the file path where the dataset is located. Afterwards, to process the dataset, the team utilized Keras' ImageDataGenerator. Parameters such as ranges, horizontal flip, and fill mode were also adjusted. Further details on the parameter inputs can be found on the Github repository.

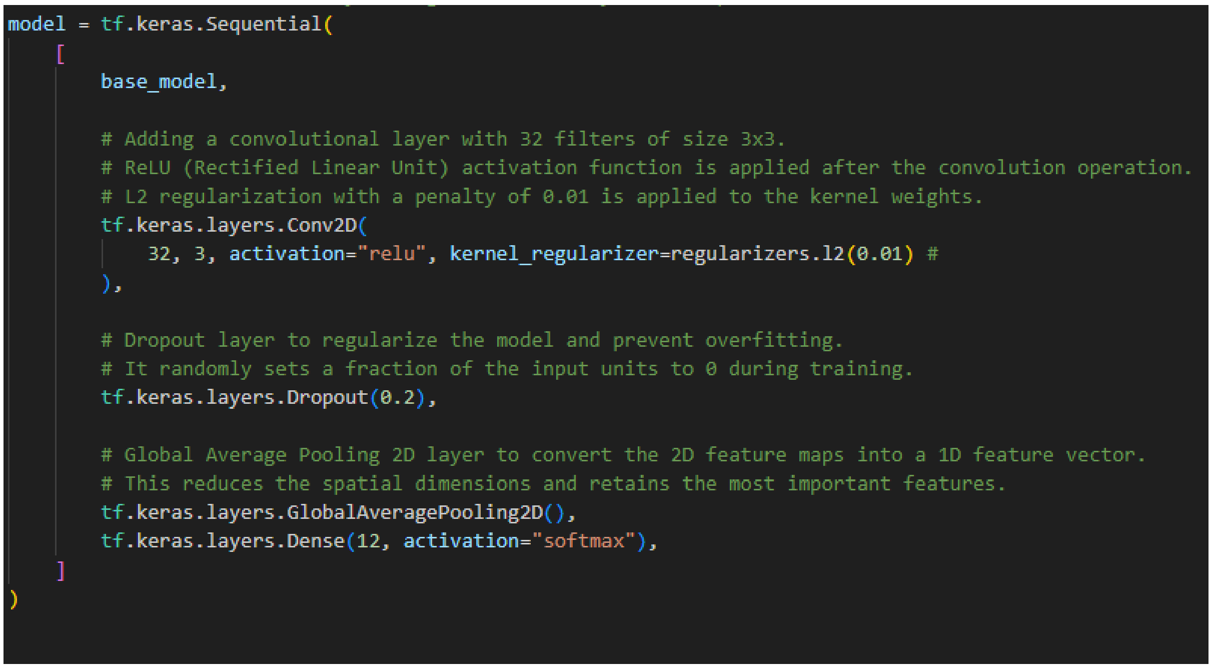
Below are the references to the Kaggle datasets that we combine to achieve a wider variety of data:

- *Fruits fresh and rotten for classification* from Sriram Reddy Kalluri
- *Fresh and Rotten Classification* from Swoyam Siddharth Nayak
- *FruitQ dataset* from Olusola Abayomi-alli
- *food freshness* from Alinesellwia
- *classification-fruits* from NGUYỄN PHAN MINH KHANG

Model Architecture



```
base_model = tf.keras.applications.MobileNetV2(
    input_shape=IMG_SHAPE, include_top=False, weights="imagenet"
)
```



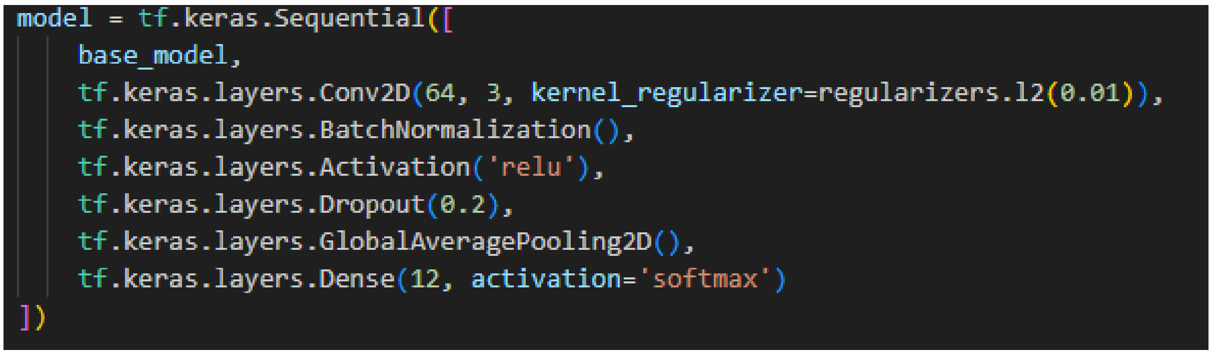
```
model = tf.keras.Sequential(
    [
        base_model,

        # Adding a convolutional layer with 32 filters of size 3x3.
        # ReLU (Rectified Linear Unit) activation function is applied after the convolution operation.
        # L2 regularization with a penalty of 0.01 is applied to the kernel weights.
        tf.keras.layers.Conv2D(
            32, 3, activation="relu", kernel_regularizer=regularizers.l2(0.01) #
        ),

        # Dropout layer to regularize the model and prevent overfitting.
        # It randomly sets a fraction of the input units to 0 during training.
        tf.keras.layers.Dropout(0.2),

        # Global Average Pooling 2D layer to convert the 2D feature maps into a 1D feature vector.
        # This reduces the spatial dimensions and retains the most important features.
        tf.keras.layers.GlobalAveragePooling2D(),
        tf.keras.layers.Dense(12, activation="softmax"),
    ]
)
```

Code Snippet of Sequential Layer (at Presentation)



```
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.Conv2D(64, 3, kernel_regularizer=regularizers.l2(0.01)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Activation('relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(12, activation='softmax')
])
```

Code Snippet of Revised Sequential Layer

MobileNetV2 is selected as the base model for transfer learning, which would be frozen to prevent the model from being trainable. This helps leverage the knowledge gained from a diverse dataset, making it easier to achieve good performance with a relatively small amount of task-specific data. In addition, it introduces the concept of inverted residuals, where the input is first expanded to a higher-dimensional space using a lightweight, depth-wise separable convolution. Then a linear bottleneck reduces the dimensionality back to the original. This structure helps capture more complex patterns efficiently. It also uses extensive depth-wise separable convolutions, which consist of a depth-wise convolution and a 1x1 point-wise convolution. This factor significantly reduces the number of parameters and computations compared to traditional convolutions. The linear bottleneck is one of the main features that ensures that the network remains computationally efficient. It involves a 1x1 convolutional layer to expand the number of channels, followed by depthwise separable convolution, and finally, another 1x1 convolutional layer to reduce the dimensionality back to the original. Additional layers are then added to the model, including the dropout layer, the global average pooling layer, and the dense output layer with softmax activation for multi-class classification. One important revision done at the end was adding batch normalization and increasing the filter size, which should theoretically help the model generalize better.

Model Compilation

The model is compiled with the Adam optimizer, categorical cross-entropy as a function, and accuracy as a metric. The learning rate was not adjusted as the team finds it unnecessary due to the fact that MobileNetV2 is quite optimized already with its architecture design and normalization technique. Furthermore, many studies suggest that the default learning rate of MobileNetV2 would be sufficient, removing the need to adjust the hyperparameter.

Training

The model is trained using 80% of the dataset for 100 epochs, with an early stopping time of 10 minutes. It would later be validated using 20% of the dataset.

TensorBoard Logging

By creating a tensorboard callback instance and adding it to the list of callbacks on the model, a better visualization of the accuracy and loss between epochs can be observed. This also allows the group to determine whether the model overfits or not.

TensorFlow Lite Model and Labels

After the model has finished being trained and evaluated, it will be converted and saved as a tflite model, which will be used as an asset in the deployment of web services using FastAPI and Render (Singh, 2019). Meanwhile, the names of the folders in the dataset would be gathered and stored in a txt file as labels.

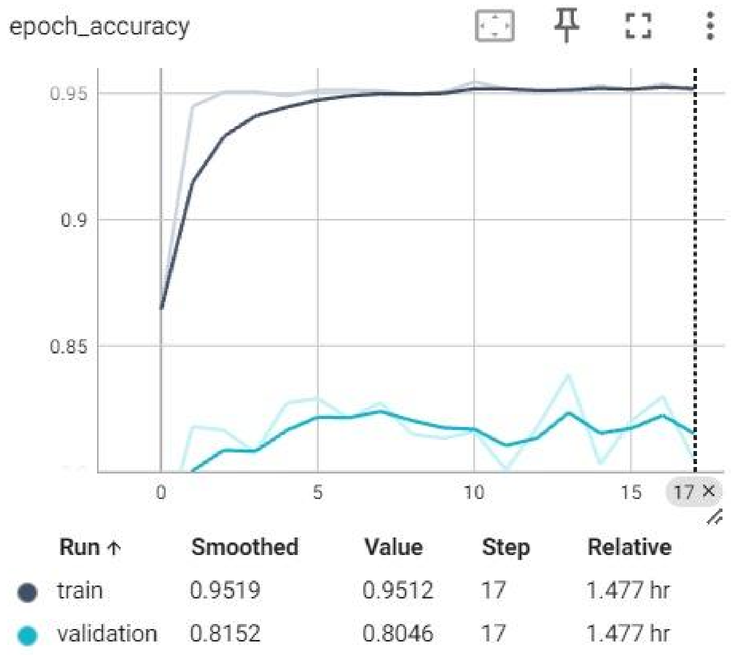
RESULTS AND DISCUSSION

The program was run on WSL using a laptop with the following specifications:

- OS: Windows 11
- CPU: AMD Ryzen 7 5800H
- Memory: 16GB, 3200MHz
- Disk: SK hynix PC711 HFS512GDE9X073N
- GPU: RTX 3070 Laptop, 8GB

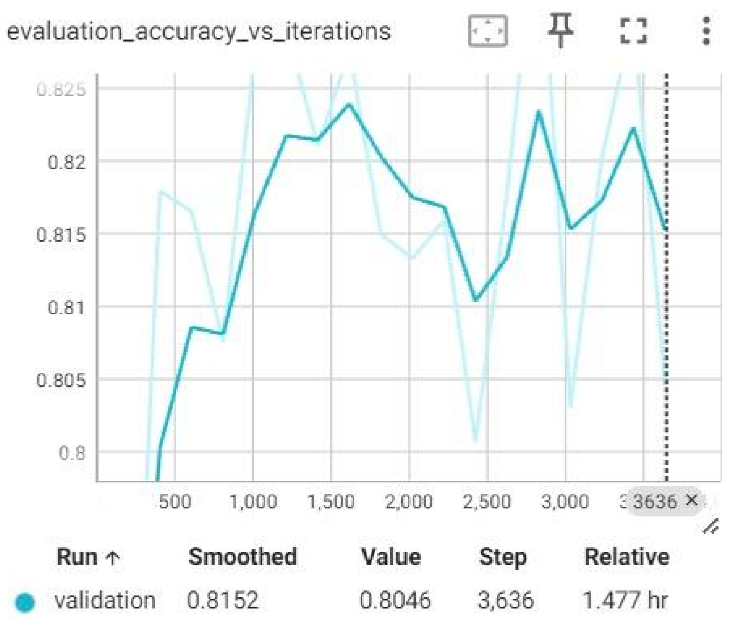
Initially, the model was planned to run for 100 epochs. However, the model stopped at the 17th epoch because an early stop was added to the model callback with a patience of 10. Here are the results of the model.

Training



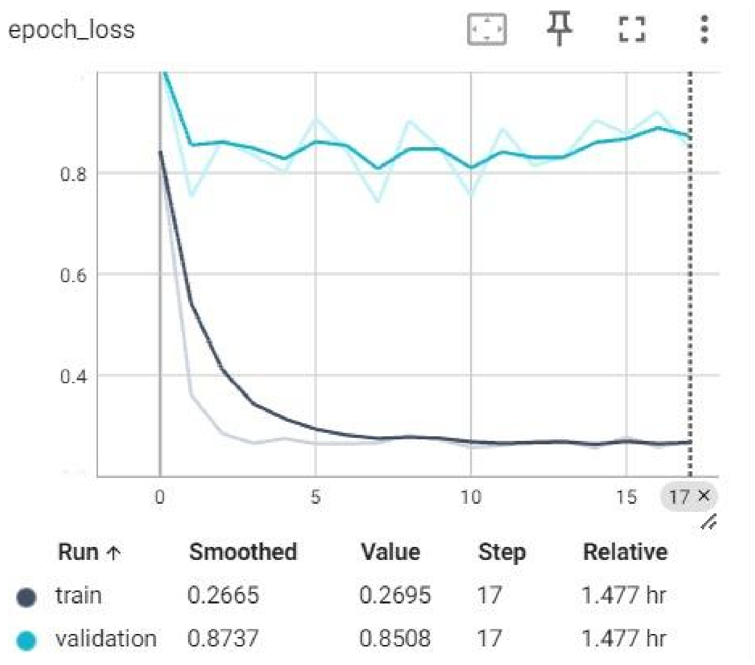
Run ↑	Smoothed	Value	Step	Relative
● train	0.9519	0.9512	17	1.477 hr
● validation	0.8152	0.8046	17	1.477 hr

The train's accuracy steadily increased with each epoch. Reaching a peak of around 0.95, which lasts from epoch 10 to 17. On the other hand, the validation accuracy hovered around 0.8 for all 17 epochs. This indicates that the model performs well on the train data, capturing and learning from the patterns. However, the fact that the training data has a significantly higher value than the validation data means that the model might be memorizing the training data's intricacies rather than generalizing well to new, unseen data from the validation set.



Run ↑	Smoothed	Value	Step	Relative
● validation	0.8152	0.8046	3,636	1.477 hr

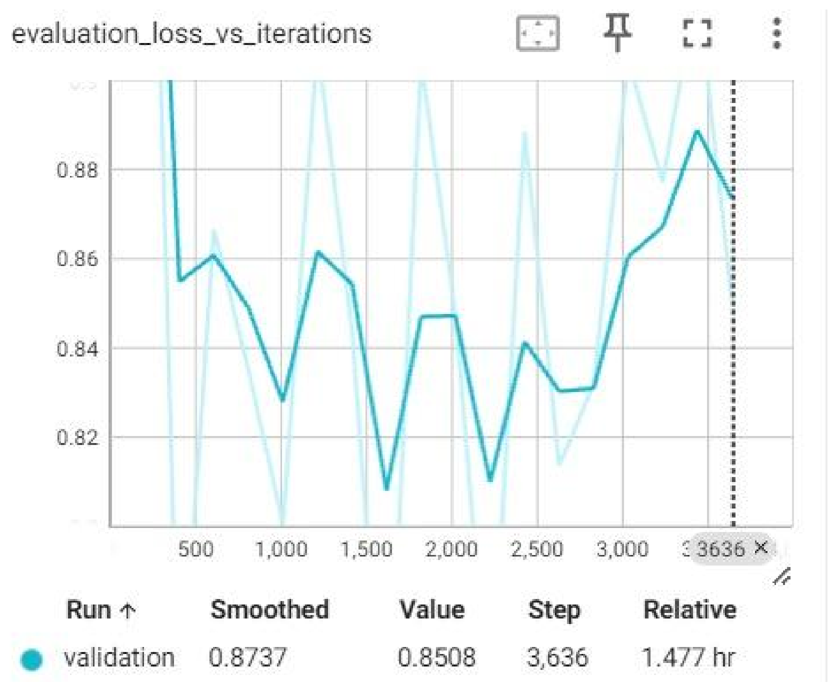
The validation of the evaluation accuracy continued to grow and reached its peak close to the validation accuracy of 0.825 at about step 1500. It went downhill towards an accuracy of 0.81 until near step 2500. However, there was a rise in validation accuracy to 0.825 at approximately step 2750. At the end, stopping at step 3636, the validation accuracy is 0.8152.



Run ↑	Smoothed	Value	Step	Relative
● train	0.2665	0.2695	17	1.477 hr
● validation	0.8737	0.8508	17	1.477 hr

The train loss consistently dropped throughout each epoch. It started at 0.8 and ended at 0.2695. This indicates that the model aligns closer to the actual labels over time.

The validation loss hovered above 0.8 at all epochs. This means that the model's predictions on the validation set consistently deviated from the true labels.



Run ↑	Smoothed	Value	Step	Relative
● validation	0.8737	0.8508	3,636	1.477 hr

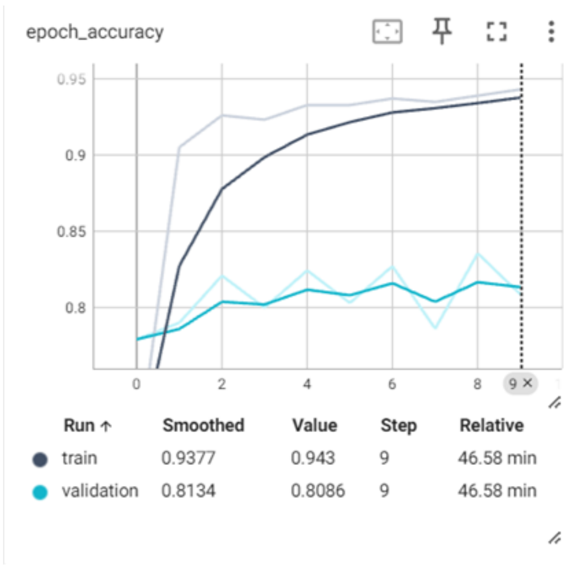
On the other hand, the validation on evaluation loss started at 1.00 and dropped to around 0.86 on step 500. It then consistently goes up and down, reaching a peak point at 0.89 on step 3400 and its lowest point at 0.8 on step 1500. It finally ends at 0.8737 on step 3636.

Tests

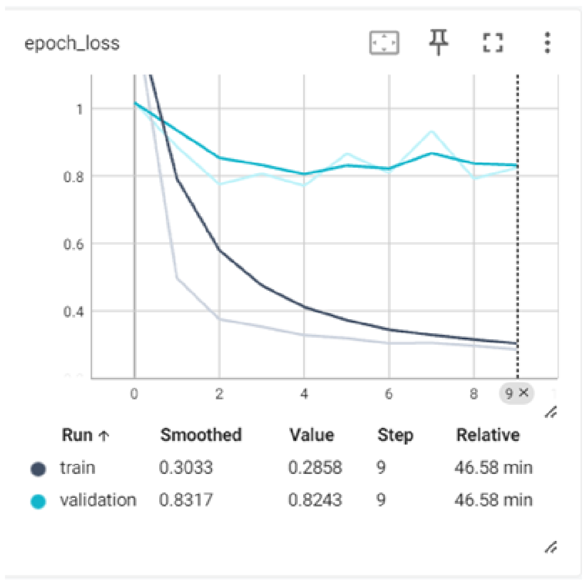
Two tests were taken: before presentation and after presentation, with modifications made to the model architecture.

Test 1: Presentation

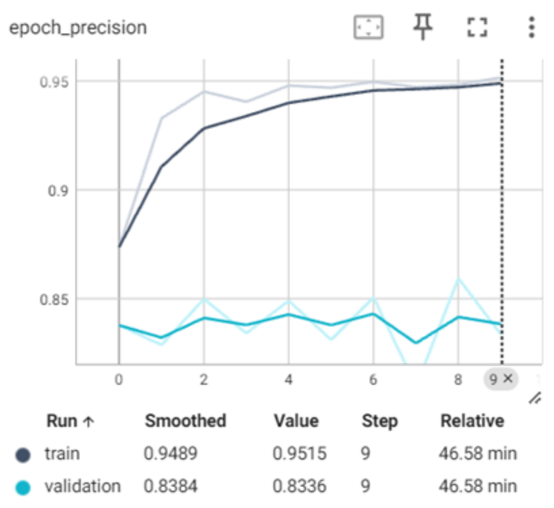
To save time, the team created a 10-epoch version of the model, which received the following training and validation status:. The results are displayed using Tensorboard for better visualization.



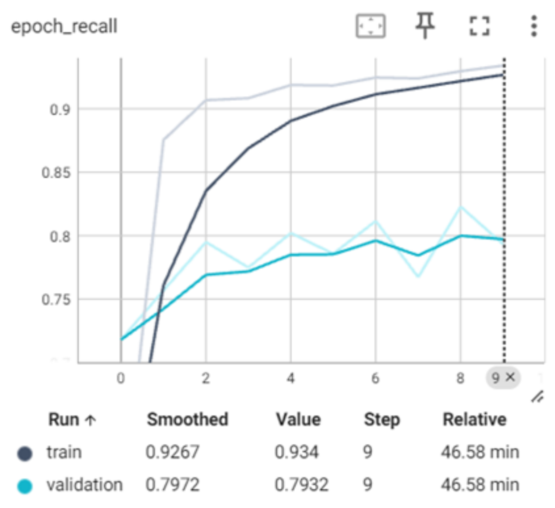
Run ↑	Smoothed	Value	Step	Relative
● train	0.9377	0.943	9	46.58 min
● validation	0.8134	0.8086	9	46.58 min



Run ↑	Smoothed	Value	Step	Relative
● train	0.3033	0.2858	9	46.58 min
● validation	0.8317	0.8243	9	46.58 min



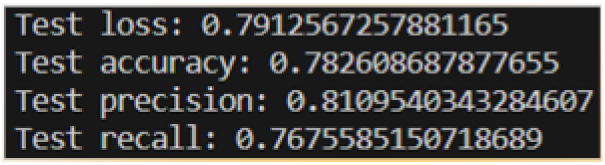
	Run ↑	Smoothed	Value	Step	Relative
●	train	0.9489	0.9515	9	46.58 min
●	validation	0.8384	0.8336	9	46.58 min



	Run ↑	Smoothed	Value	Step	Relative
●	train	0.9267	0.934	9	46.58 min
●	validation	0.7972	0.7932	9	46.58 min

As seen from the images above, the model was still able to perform well on the validation set, similar to the 17-epoch version of the model.

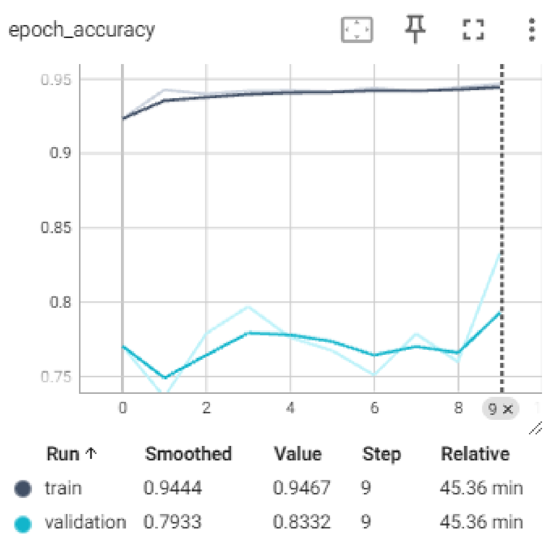
Afterwards, the team evaluated using a separate dataset and obtained the following results. Based on the observation, it can be seen that the model has performed well, as there is no significant difference between the results obtained from validation and testing.



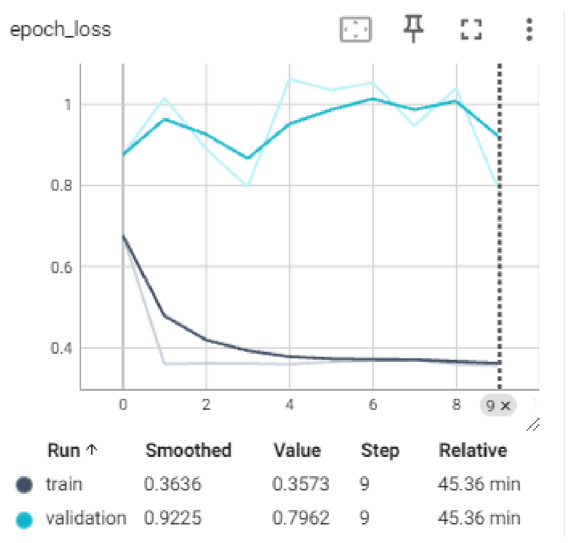
```
Test loss: 0.7912567257881165
Test accuracy: 0.782608687877655
Test precision: 0.8109540343284607
Test recall: 0.7675585150718689
```

Test 2 : Revision

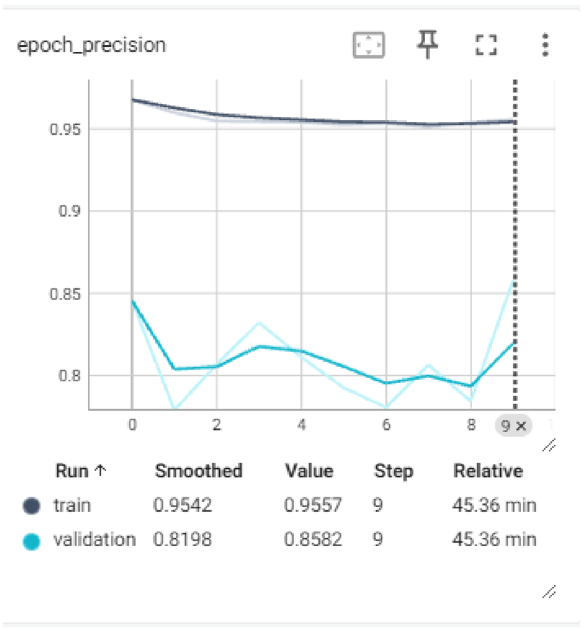
As of the revision of the report, there has been a modification in the model architecture in hopes of better generalization. The changes were in filter size from 32 to 64 and the inclusion of batch normalization. For clarification, please refer to the code snippets in the Model Architecture section. Similar to the previous test, the team also utilized 10 epochs for maximum time efficiency.



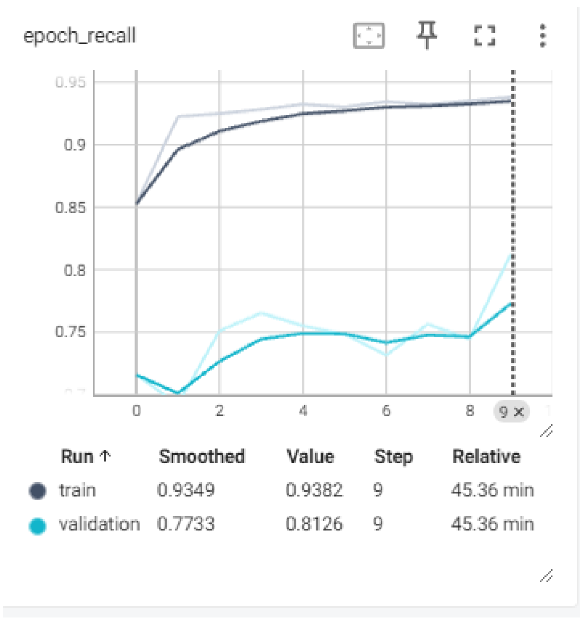
	Run ↑	Smoothed	Value	Step	Relative
●	train	0.9444	0.9467	9	45.36 min
●	validation	0.7933	0.8332	9	45.36 min



	Run ↑	Smoothed	Value	Step	Relative
●	train	0.3636	0.3573	9	45.36 min
●	validation	0.9225	0.7962	9	45.36 min

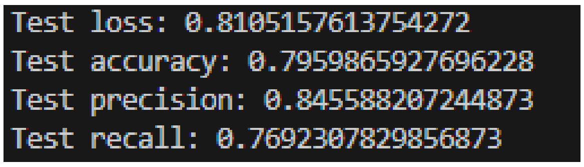


Run ↑	Smoothed	Value	Step	Relative
● train	0.9542	0.9557	9	45.36 min
● validation	0.8198	0.8582	9	45.36 min



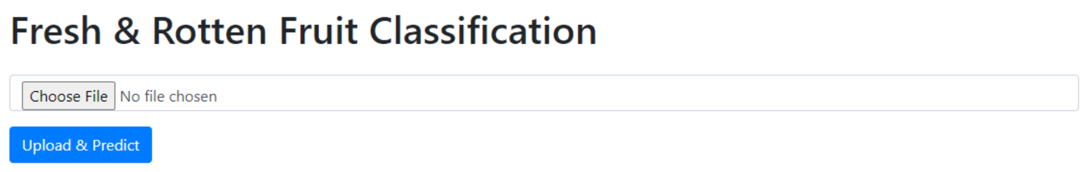
Run ↑	Smoothed	Value	Step	Relative
● train	0.9349	0.9382	9	45.36 min
● validation	0.7733	0.8126	9	45.36 min

Upon viewing the results on the terminal, there don't appear to be any considerable changes in the test result when compared with the first test results.

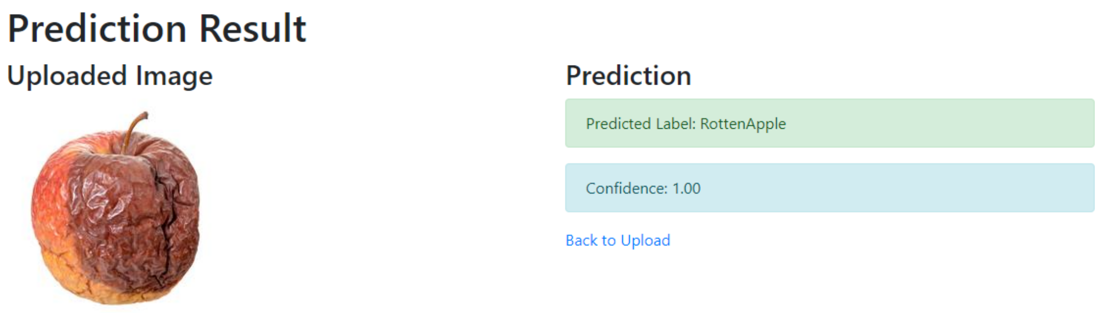

```
Test loss: 0.8105157613754272
Test accuracy: 0.7959865927696228
Test precision: 0.845588207244873
Test recall: 0.7692307829856873
```

Code Snippet of Test Result (Revision)

User Interface


Fresh & Rotten Fruit Classification

The home page interface provides a user-friendly platform for users to interact with the model. The design is intuitive, allowing users to easily upload images for classification. On mobile phones, users can take pictures or upload a picture before predicting the fruit classification.


Prediction Result

Uploaded Image

Prediction

Predicted Label: RottenApple

Confidence: 1.00

Upon uploading an image, the model rapidly processes it and returns the predicted classification along with a confidence score. This streamlined process ensures a seamless user experience. For now, there are 13 possible predicted label outputs: FreshApple, RottenApple, FreshOrange, RottenOrange, FreshMango, RottenMango, FreshBanana, RottenBanana, FreshTomato, RottenTomato, FreshStrawberry, RottenStrawberry, and unknown fruit.

The confidence score (0.00 - 1.00) serves as a quantitative measure that accompanies the predicted classification output of the model. Essentially, it represents the model's level of certainty in its prediction. A higher confidence score indicates that the model is more confident in its classification decision, whereas a lower score suggests a degree of uncertainty. The current confidence score threshold is 0.8.

REFERENCES

Ahmed, I. (2023, July 1). Why your MobileNetv2 model performs better than Resnet50. Medium. https://medium.com/@imtiaz.ahmed2206/why-your-mobilenetv2-model-performs-better-than-resnet50-2a9998fda4c7

Aline Sellwia, N. (n.d.). Food Freshness. Kaggle. Retrieved from https://www.kaggle.com/datasets/alinesellwia/food-freshness

FastAPI. (n.d.). FastAPI web framework. Retrieved from https://fastapi.tiangolo.com/

Minh Khang, N. (n.d.). Classification Fruits. Kaggle. Retrieved from https://www.kaggle.com/datasets/nguynphanminhkhang/classification-fruits

Ramaswamy, S. (n.d.). Fruits Fresh and Rotten for Classification. Kaggle. Retrieved from https://www.kaggle.com/datasets/sriramr/fruits-fresh-and-rotten-for-classification

Sholzz. (n.d.). FruitQ Dataset. Kaggle. Retrieved from https://www.kaggle.com/datasets/sholzz/fruitq-dataset

Singh, A. & Bhadanim, R. . (2019). Mobile Deep Learning with TensorFlow Lite, ML Kit, and Flutter. Packt Publishing. Retrieved from https://www.packtpub.com/product/mobile-deep-learning-with-tensorflow-lite-ml-kit-and-flutter/9781789611212

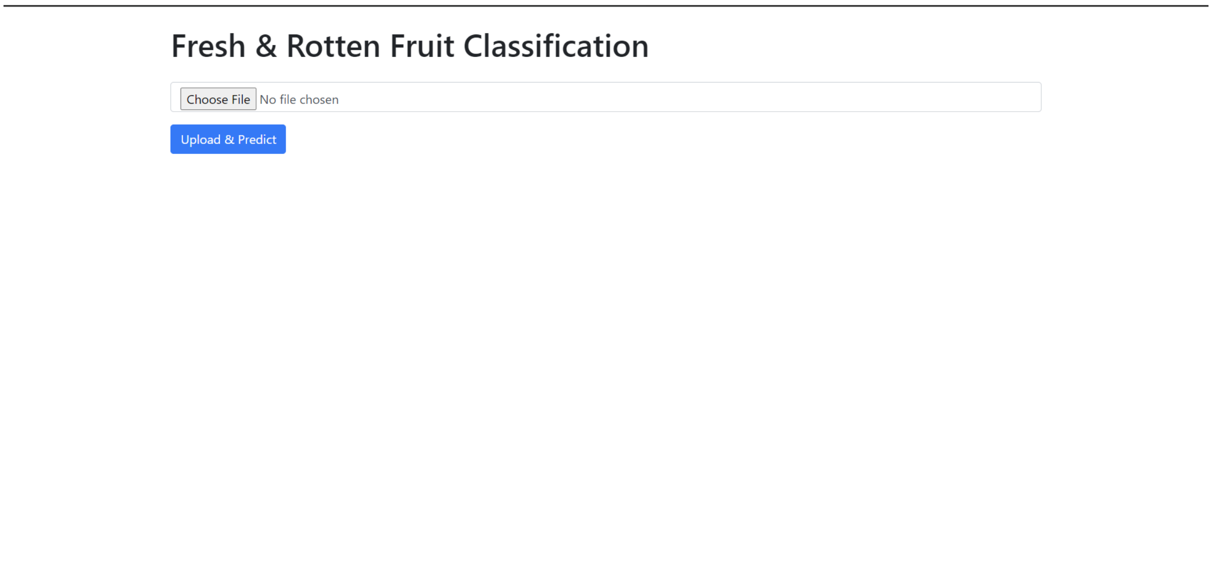
Swoyam2609. (n.d.). Fresh and Stale Classification. Kaggle. Retrieved from https://www.kaggle.com/datasets/swoyam2609/fresh-and-stale-classification

TechTarget. (n.d.). Bootstrap. In A. Zola (Ed.), WhatIs.com. Retrieved from https://www.techtarget.com/whatis/definition/bootstrap

APPENDICES

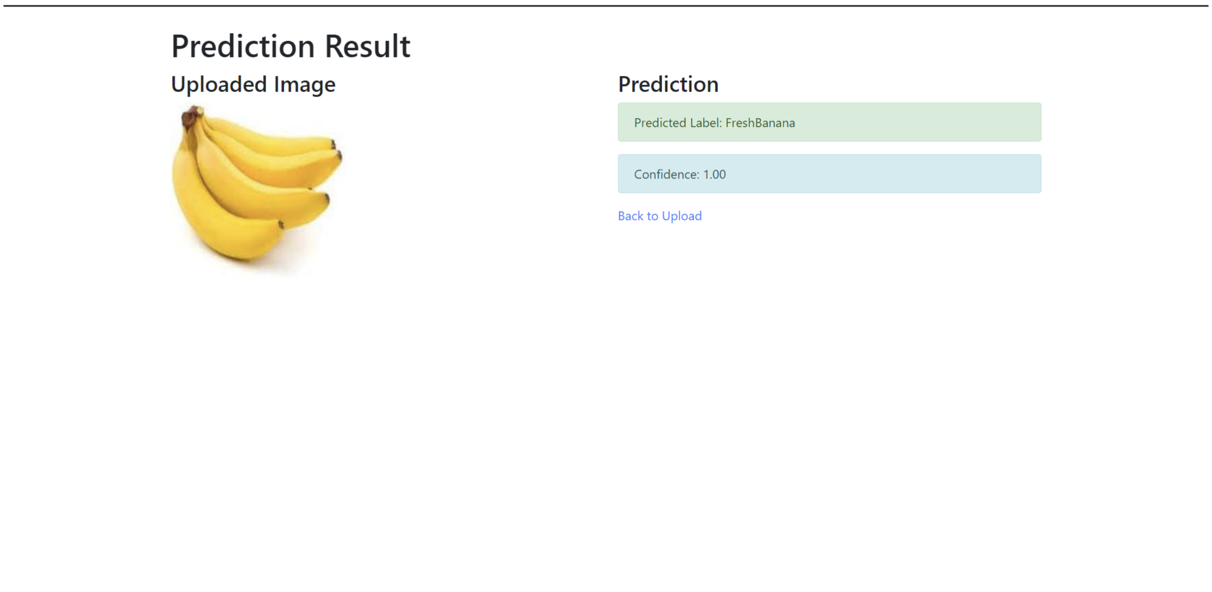
Program Manual

1. User input this link on browser https://fresh-rotten-fruits-classifier.onrender.com/
2. After page loads, it will display this screen



Fresh & Rotten Fruit Classification

3. Choose image to input for prediction. For users on mobile devices like Android smartphones, they can opt to take photos via Camera app.
4. Click the “Upload and Predict” button.
5. After clicking the button, the model will process the input image and bring the user to the /predict page



Prediction Result

Uploaded Image

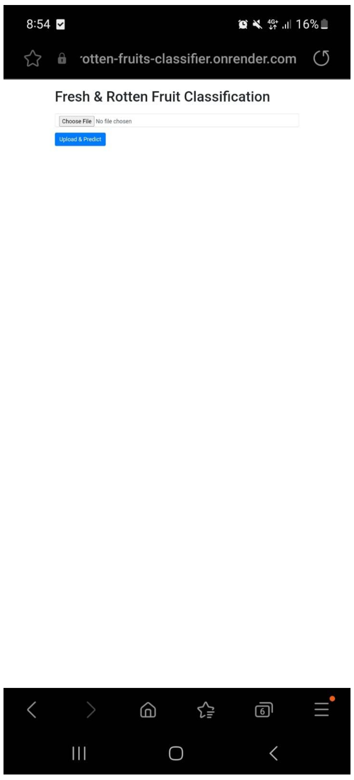
Prediction

Predicted Label: FreshBanana

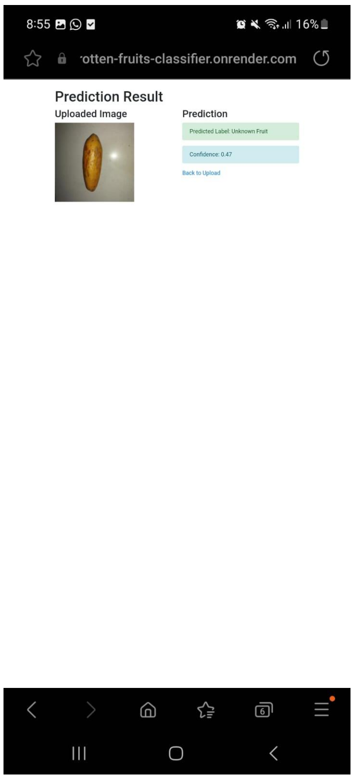
Confidence: 1.00

6. On the predict page, users can see the predicted label of the input image and the confidence rate for the image prediction.

7. If user wants to repeat process, they can click the “Back to Upload” link



Fresh & Rotten Fruit Classification



Prediction Result

Uploaded Image

Prediction

Predicted Label: Unknown Fruit

Confidence: 0.47

Image 1: Phone View of Index Page (Left)   *Image 2: Phone View of Predict Page (Right)*

Video Demo

https://youtu.be/ZEOwV2rRGKM

GitHub Link

https://github.com/Theotrgl/AI-Sem5-Fruit-Classification

Canva Link

https://www.canva.com/design/DAF2LmEBi_4/VaSDTjVEbfaYYpDx4duGhA/edit?utm_content=DAF2LmEBi_4&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton