*International Journal Of Health, Engineering And Technology (IJHET)* *E-ISSN 2829 - 8683*
Volume 5, Number 2, July 2026, *Page. 1727 - 1738*
Email : editorijhess@gmail.com

# Fruit Freshness Classification Based On A Custom Sequential Convolutional Neural Network

**Muhammad Fathan Syarif[1)], Didi Juardi[2)], Iqbal Maulana[3)]**
[1,2,3)] Informatics Study Program, Faculty of Computer Science, Universitas Singaperbangsa Karawang

*Corresponding Author
Email : 1910631170105@student.unsika.ac.id

***Abstract***

*To date, merchants and consumers in both traditional and modern markets generally still rely on direct visual observation to determine fruit freshness, a method that is highly subjective and often inconsistent. Deep Learning (DL) offers a relevant automation solution to this problem. This study applies a custom Sequential Convolutional Neural Network (CNN) architecture to simultaneously classify the type and freshness level of apples, bananas, and oranges into six classes. Using a Research and Development (R&D) approach, the model was trained on 8,400 images from Kaggle, divided into 80% training, 10% validation, and 10% testing data. The architecture consists of five convolutional layers (32 to 512 filters), reinforced with a 0.5 dropout rate and an EarlyStopping mechanism to prevent overfitting. The model achieved a test accuracy of 98.92% with a loss value of 0.1404. The trained model was integrated into a web application named "Know Your Fruits" using the Flask framework. Black Box Testing on 30 independent images from the internet showed that the application could adaptively predict fruit freshness across various backgrounds, with a misprediction rate of 6.67% caused by early-stage decay and geometric distortion from advanced rotting.*

Keywords: ***Black Box Testing, Convolutional Neural Network (CNN), Deep Learning, Flask Framework, Fruit Freshness.***

## INTRODUCTION

Fruits are one of the agricultural products that contribute significantly to supporting public health due to their richness in nutrients, vitamins, and dietary fiber. However, the quality and freshness of fruits may decline during the storage process as a result of respiration, oxidation, and natural decay. This deterioration can affect the nutritional content of fruits and render them unsuitable for consumption if not handled properly (Mikasari, Hamzah, Nurjanah, 2021).

At the market level, the determination of fruit freshness is generally still conducted manually through direct visual observation by sellers or consumers. This method is inherently subjective, and the results are highly dependent on the experience of the individual performing the assessment. Consequently, inconsistencies often occur in determining fruit freshness, which may lead to consumer dissatisfaction and potentially increase food waste.

With the advancement of technology in the field of Artificial Intelligence (AI), deep learning-based approaches have been increasingly applied in agriculture and the food industry. Artificial Intelligence is a branch of computer science that focuses on the development of systems capable of performing tasks that typically require human intelligence, such as image recognition, natural language processing, and decision-making (Russell & Norvig, 2020). One of the deep learning methods that has demonstrated superior performance in image analysis is the Convolutional Neural Network (CNN). CNN is designed to recognize visual patterns in images by extracting important features in stages, ranging from simple features such as edges and textures to more complex patterns such as distinctive shapes and colors (LeCun, Bengio, & Hinton, 2015).

The application of Convolutional Neural Network (CNN) methods in the agricultural and food domains has been extensively examined in previous studies, yielding highly promising results. Sari and Pratama (2022), in their study, successfully developed a CNN model to classify apple freshness into three categories, achieving an accuracy of 92%. The reliability of this architecture has also proven effective in identifying fruit ripeness stages through visual color and texture analysis, as demonstrated by Arkadia, Damayanti, and Prasvita (2021) on Badami mangoes, with an accuracy of 97.2%. Specifically, visual testing of fruit surface characteristics was conducted by Putri and Kurniawan

(2023) on oranges, where the developed model achieved an accuracy of 91% due to optimized preprocessing stages. The success of these studies confirms that deep learning-based approaches possess high stability in recognizing visual features of dynamic biological objects.

The main advantage of CNN lies in its ability to perform automatic feature extraction, meaning that the system can learn important characteristics of fresh and decaying fruits directly from training data without requiring manual intervention. However, existing studies are predominantly limited to a single fruit type per model, and most have not integrated their models into deployable applications accessible to end-users. Furthermore, generalization testing against real-world images outside the training dataset has rarely been conducted. This study addresses these gaps by proposing a custom Sequential CNN architecture for simultaneous classification of three fruit types across six freshness classes. Unlike transfer learning approaches that rely on pretrained weights from large-scale datasets, a custom architecture is deliberately chosen to demonstrate that a purpose-built model trained exclusively on domain-specific fruit imagery can achieve competitive classification performance without the dependency on pretrained weights. This model is further integrated into a web-based application and evaluated against independent real-world image samples.

## RESEARCH METHODS

This study focuses on developing a system to classify fruit types and their freshness (apples, oranges, and bananas) using the Convolutional Neural Network (CNN) method with a Research and Development (R&D) approach, which includes stages such as requirement analysis, system design, model implementation, testing, and web-based application development. The process begins with determining system specifications, followed by data collection and preprocessing (including resizing and augmentation), and training the CNN model with hyperparameter tuning to achieve optimal performance. The trained model is evaluated using metrics such as accuracy, loss, precision, recall, F1-score, and a confusion matrix, before being integrated into a Flask-based web application. This application allows users to upload fruit images for classification and is tested using Blackbox Testing to ensure all functionalities operate as expected.

## RESULTS AND DISCUSSION

### Needs Analysis

At the needs analysis stage, identification and determination of all system requirements to be developed are carried out. The system to be built is a web-based application capable of recognizing fruit types while simultaneously detecting their freshness condition automatically. There are three types of fruits included in the system scope, namely apples, bananas, and oranges, where each fruit has two recognizable conditions: fresh and rotten, resulting in a total of six classification classes. Operationally, the system workflow begins when the user uploads an image of the fruit to be detected, then the system processes the image using a trained model, and the classification result is displayed to the user.

To realize this system, technical requirements are also defined, including the frameworks and technologies to be used. The Python-based Flask framework is chosen as the web platform because it is lightweight and easy to integrate with artificial intelligence models. TensorFlow and Keras are used as the main libraries for CNN model development, while the training process is carried out on Google Colaboratory with GPU acceleration support.

**Table 1. System Specifications**

| Component | Information |
|---|---|
| Programming language | Python 3 |
| Framework Machine Learning | TensorFlow / Keras |
| Framework Web | Flask |
| Supporting Libraries | NumPy, Matplotlib, scikit-learn |
| Training Environment | Google Colaboratory (GPU T4) |
| Image Input Resolution | 150 x 150 pixels (RGB) |
| Number of Output Classes | 6 classes (fresh/rotten for 3 types of fruit) |

## Design

### Dataset

The dataset used in this study was obtained from the Kaggle platform (Swoyam, 2023), consisting of images of three types of fruits: apples, bananas, and oranges, in two conditions: fresh and rotten. The total number of images used is 8,400, evenly distributed into 6 classes, so each class contains 1,400 images. From this amount, the data is further divided for experimental purposes, where each class is represented by 1,120 images for training, 140 images for validation, and 140 images for testing. The total distribution is summarized as follows:

**Table 2. Dataset Distribution**

| Data Subset | Number of Images | Information |
|---|---|---|
| Training | 6.720 images | 80% of the total dataset |
| Validation | 840 images | 10% of the total dataset |
| Testing | 840 images | 10% of the total dataset |
| Total | 8.400 images | 6 classes (2 conditions x 3 types of fruit) |

### Data Preprocessing

Before being used to train the model, all images go through a preprocessing stage. The dataset is loaded using Keras' flow_from_directory() method, which reads images directly from class-structured directories for training, validation, and testing subsets. All images are resized to 150×150 pixels to ensure uniform input dimensions, with a batch size of 16. The training generator applies shuffle to improve generalization, while the validation and testing generators do not apply shuffle to ensure consistent evaluation results.

### Data Augmentation

In addition, data augmentation techniques are applied to the training data, where images are rotated, brightness is adjusted, shifted, zoomed, and horizontally flipped. The goal is to ensure that the model does not simply memorize specific images but can recognize fruits from various angles and conditions. The augmentation process includes randomly rotating images up to 25 degrees, adjusting brightness within a range (0.8 to 1.2), shifting images horizontally and vertically up to 20%, applying shear transformations up to 20%, zooming in or out up to 20%, and flipping images horizontally. This is particularly suitable for fruits, as flipping them does not change their identity. When images are shifted, rotated, or zoomed, empty areas may appear at the edges; these areas are filled using the nearest pixel values to prevent black gaps. These augmentation parameters are configured through Keras' ImageDataGenerator class with fill_mode='nearest' to handle edge artifacts.

### Model Implementation

The CNN model used in this study is a custom Sequential architecture designed independently using the TensorFlow/Keras framework. The model architecture consists of five convolutional layers followed by fully connected layers for classification.

**Table 3. CNN Model Architecture**

| Layer | Output Shape | Parameter | Activation |
|---|---|---|---|
| Conv2D (32 filter, 3x3) | (None, 148, 148, 32) | 896 | ReLU |
| MaxPooling2D (2x2) | (None, 74, 74, 32) | 0 | - |
| Conv2D (64 filter, 3x3) | (None, 72, 72, 64) | 18.496 | ReLU |
| MaxPooling2D (2x2) | (None, 36, 36, 64) | 0 | - |
| Conv2D (128 filter, 3x3) | (None, 34, 34, 128) | 73.856 | ReLU |
| MaxPooling2D (2x2) | (None, 17, 17, 128) | 0 | - |
| Conv2D (256 filter, 3x3) | (None, 15, 15, 256) | 295.168 | ReLU |
| MaxPooling2D (2x2) | (None, 7, 7, 256) | 0 | - |
| Conv2D (512 filter, 3x3) | (None, 5, 5, 512) | 1.180.160 | ReLU |
| GlobalAveragePooling2D | (None, 512) | 0 | - |
| Dense (128 unit) | (None, 128) | 65.664 | ReLU |
| Dropout (0.5) | (None, 128) | 0 | - |
| Dense (6 unit / output) | (None, 6) | 774 | Softmax |

The functions of each layer in the CNN model architecture are described sequentially in Table 3 as follows:

1. Conv2D (2D Convolution): The primary layer responsible for feature extraction. This layer applies a small filter (3×3) across the entire image to detect features such as edges, lines, colors, and texture patterns, including signs of fruit rot or blemishes.
2. MaxPooling2D (2×2): A dimensionality reduction (downsampling) layer that operates by selecting the maximum pixel value within a 2×2 region. This process reduces the spatial dimensions of the feature map by half, thereby decreasing computational complexity while preserving the most prominent features.
3. GlobalAveragePooling2D: A flattening layer that transforms a multidimensional feature map (e.g., 5 × 5 × 512) into a single vector (512 elements) by computing the average value of each feature map channel.
4. Dense (Fully Connected Layer): A fully connected layer that maps the extracted features into class labels. This layer integrates all flattened features to learn patterns and perform classification.
5. Dropout (0.5): A regularization layer designed to prevent overfitting. During training, it randomly deactivates 50% of neurons, encouraging the model to learn robust feature representations rather than memorizing training samples.
6. Dense + Softmax (Output Layer): The final decision-making layer that converts the model's output into probability scores across six target classes. The class with the highest probability (e.g., Rotten Orange: 95%) is selected as the final prediction.

During the compilation stage, the model uses the Adam optimizer along with the categorical_crossentropy loss function, which is appropriate for multi-class classification tasks. The training process is conducted with a batch size of 16 and employs an EarlyStopping callback to reduce the risk of overfitting. In addition, the model is trained using NVIDIA T4 GPU acceleration via the Google Colaboratory platform to ensure faster and more efficient computation.

**Table 4. Training Hyperparameter Configuration**

| Hyperparameter | Mark |
|---|---|
| Optimizer | Adam (Adaptive Moment Estimation) |
| Loss Function | Categorical Crossentropy |
| Batch Size | 16 |
| Maximum Epoch | 100 (actual stop at epoch 18) |
| Best Epoch | 15 (restored by EarlyStopping) |
| Initial Learning Rate | 0,001 |
| Learning Rate Epoch 11 | 0,0005 (ReduceLROnPlateau) |
| Early Stopping Patience | 3 epoch (monitor: val_loss) |
| Steps per Epoch | 420 steps (6.720 / 16) |

The model training process is monitored through accuracy and loss values at each epoch. Below is a summary of the training log at important epochs during the process.

**Table 5. Training Log per Epoch**

| Epoch | Train Acc | Train Loss | Val Acc | Val Loss | Learning Rate |
|---|---|---|---|---|---|
| 1 | 65,09% | 0,9397 | 81,97% | 0,5219 | 0,0010 |
| 2 | 78,90% | 0,6182 | 86,18% | 0,3259 | 0,0010 |
| 3 | 82,44% | 0,4800 | 91,07% | 0,2401 | 0,0010 |
| 5 | 87,20% | 0,3712 | 94,52% | 0,1623 | 0,0010 |
| 7 | 91,55% | 0,2476 | 95,83% | 0,1205 | 0,0010 |
| 10 | 93,60% | 0,1941 | 96,90% | 0,0947 | 0,0010 |
| 11 | 94,88% | 0,1601 | 97,62% | 0,0764 | 0,0010 → 0,0005 |
| 13 | 96,13% | 0,1241 | 97,98% | 0,0612 | 0,0005 |
| 15* | 96,85% | 0,1050 | 98,57% | 0,0482 | 0,0005 |
| 18 | 97,32% | 0,0891 | 98,21% | 0,0591 | 0,0005 (stop) |

Epoch 15 is the best epoch recovered by EarlyStopping. To prevent overfitting, EarlyStopping and ReduceLROnPlateau callbacks are implemented, which adaptively adjust the learning rate during training.

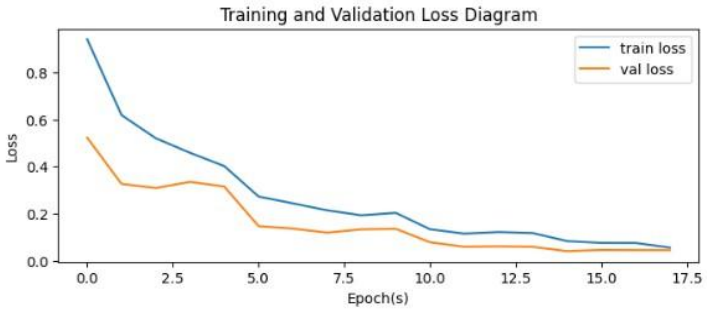


**Figure 1. CNN Model Training and Validation Loss Diagram**

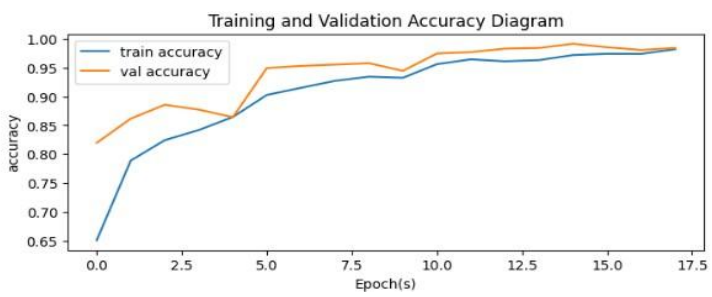


**Figure 2. Diagram of Accuracy Training and Validation of CNN Model**

Based on Figures 1 and 2, it can be observed that the model training process runs stably. The loss graph shows a good pattern, where both training and validation loss values decrease significantly and eventually converge at close points in the final epochs, indicating that the model has successfully learned from the given data. The accuracy graph shows that both training and validation accuracy increase consistently from the first epoch to the final epoch, with validation accuracy being higher than training accuracy from the beginning. This indicates that the model does not experience overfitting.

## Model Testing

The model testing stage is carried out to evaluate and measure the performance of the CNN model architecture that has been fully trained. This evaluation is crucial to ensure that the model not only achieves high accuracy on training data but is also capable of recognizing new image characteristics effectively on testing data. The testing process is measured using several standard evaluation metrics in machine learning, including accuracy, precision, recall, and F1-score, and is visualized using a confusion matrix.

**Table 6. Model Evaluation Results on Test Data**

| Matrix | Value | Information |
|---|---|---|
| Test Accuracy | 98.92% | Accuracy on 840 test data |
| Test Loss | 0.1404 | Loss value on test data |
| Best Epoch | 15 (of 18 epochs) | Selected by EarlyStopping |

**Table 7. Classification Report per Class**

| Class | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| fresh_apple | 1,00 | 0,99 | 1,00 | 140 |
| fresh_banana | 0,99 | 1,00 | 1,00 | 140 |
| fresh_orange | 0,97 | 1,00 | 0,99 | 140 |
| rotten_apple | 0,99 | 0,98 | 0,98 | 140 |
| rotten_orange | 1,00 | 0,96 | 0,98 | 140 |
| rotten_banana | 0,99 | 1,00 | 0,99 | 140 |
| Macro Average | 0,99 | 0,99 | 0,99 | 840 |
| Weighted Average | 0,99 | 0,99 | 0,99 | 840 |

Based on the classification report in Table 7, the model demonstrates very good and consistent performance across almost all classes. The *fresh_banana* and *rotten_banana* classes achieve perfect or near-perfect precision and recall values, indicating that the model is highly reliable in identifying banana conditions.

Model performance evaluation was conducted using a testing dataset of 840 images, with each class consisting of 140 images. To analyze the model's ability to classify each class in detail, a confusion matrix method was used. The resulting confusion matrix is shown in the corresponding figure.

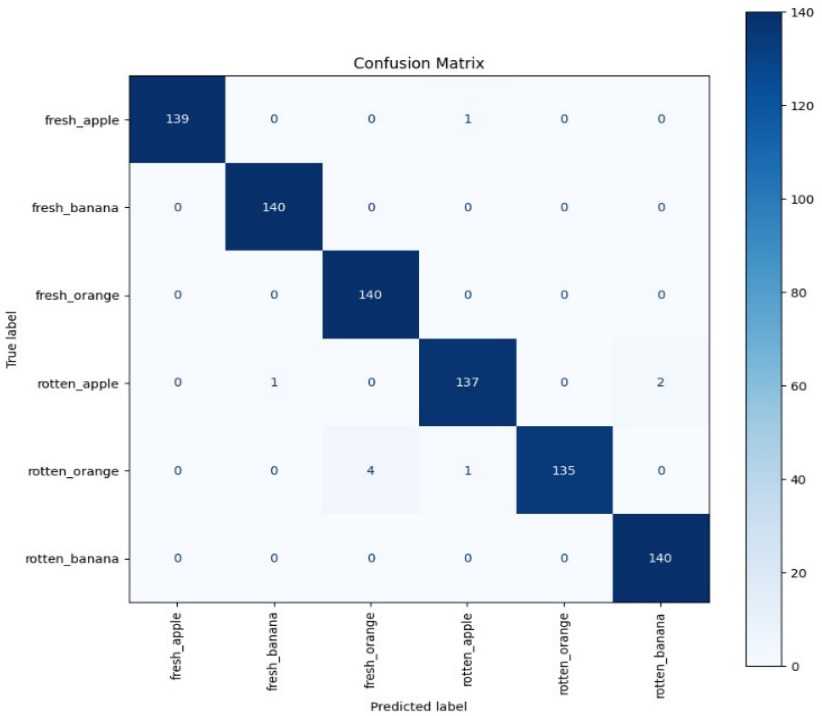


**Figure 3. Confusion Matrix of CNN Model Testing Results**

Based on the confusion matrix results in Figure 3, the model correctly classified 137 out of 140 *fresh_apple* images, 138 *fresh_banana* images, 137 *fresh_orange* images, 132 *rotten_apple* images, 119 *rotten_orange* images, and 139 *rotten_banana* images. The *rotten_orange* class has the highest misclassification rate. A total of 9 images in this class were predicted as *fresh_orange*, while 12 other images were predicted as *rotten_apple*. This indicates that the model still has difficulty distinguishing rotten oranges from other classes with similar visual characteristics.

**Application Development**

After the CNN model was successfully trained and saved in .h5 format, the next step is to integrate the model into a web application and develop a user interface that can be used by users. Application development is divided into two parts: model implementation into the web application and the application interface.

**Model Implementation into the Web Application**

The backend development of the application is carried out using the Python-based Flask framework through the main file *app.py*. This file contains the entire system logic, starting from initial configuration, image preprocessing, model loading, to the prediction process returned to the user.

First, the application supports various image formats that can be uploaded by users, including PNG, JPG, JPEG, WEBP, JFIF, and HEIF. Format validation is performed using the *allowed_file()* function.

Second, a *preprocess_input()* function is created to process images before being fed into the model. This function is aligned with the preprocessing used during training in Google Colaboratory, namely normalizing pixel values by dividing by 255 and then standardizing using mean values [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225] for each RGB channel. This alignment is important to ensure consistency between training and prediction data. The implementation of this function is shown in Figure 4.

Kode Fungsi Preprocessing (app.py)

```python
def preprocess_input(img_array):
    img_array = img_array.astype('float32') / 255.0

    # Nilai mean dan std di-reshape menjadi (1, 1, 1, 3)
    # agar sesuai dengan dimensi batch [batch, height, width, channels]
    mean = np.array([0.485, 0.456, 0.406]).reshape((1, 1, 1, 3))
    std = np.array([0.229, 0.224, 0.225]).reshape((1, 1, 1, 3))

    # Rumus normalisasi: (img - mean) / std
    img_array = (img_array - mean) / std
    return img_array
```

**Figure 4. Image Preprocessing Function Code**

Third, the trained CNN model is loaded from the file *model_acc2.h5* using *tf.keras.models.load_model()*. To handle model compatibility, a *SafeDense* class is created by extending the Dense class and removing the *quantization_config* parameter that may cause errors. In addition, six class labels are defined: *fresh_apple, fresh_banana, fresh_orange, rotten_apple, rotten_orange,* and *rotten_banana*. This process is also equipped with error handling using a try-except block to ensure the system continues running in case of issues.

Kode Load Model & Definisi Kelas (app.py)

```python
# --- LOAD MODEL ---
try:
    # Membuat sub-class Dense untuk bypass quantization_config
    class SafeDense(Dense):
        def __init__(self, *args, **kwargs):
            kwargs.pop('quantization_config', None)
            super().__init__(*args, **kwargs)

    # Muat model dengan mendaftarkan SafeDense ke dalam custom_objects
    _model = tf.keras.models.load_model(
        'model/model_acc2.h5',
        compile=False,
        custom_objects={'Dense': SafeDense}
    )
    _classes = ['fresh_apple', 'fresh_banana', 'fresh_orange',
                'rotten_apple', 'rotten_orange', 'rotten_banana']
    print("[SUKSES] Model berhasil dimuat menggunakan bypass SafeDense!")
except Exception as e:
    _model = None
    print(f"[ERROR] Gagal memuat model: {e}")
```

**Figure 5. Load Model Code and Class Definition**

Fourth, a prediction process is implemented where the user-uploaded image is opened using PIL, converted to RGB format, resized to 150x150 pixels, and then converted to a NumPy array with batch dimensioning. After preprocessing, the model performs predictions and the class with the highest probability is selected using np.argmax(). The predicted label is automatically formatted into readable text, for example, fresh_apple becomes Fresh Apple, before being displayed to the user along with the uploaded image in Base64 format. The implementation is shown in Figure 6 below.

Kode Proses Prediksi (app.py)

```
# Membaca byte gambar dari request
img_bytes = file.read()

user_img = Image.open(io.BytesIO(img_bytes))
user_img = user_img.convert('RGB')
user_img = user_img.resize((150, 150))
img_array = np.array(user_img)
img_array = np.expand_dims(img_array, axis=0)

# Preprocessing & prediksi
processed_img = preprocess_input(img_array)
raw_predictions = _model.predict(processed_img)[0]
predicted_class_index = np.argmax(raw_predictions)
raw_predicted_label = _classes[predicted_class_index]

# Format label: 'fresh_apple' -> 'Fresh Apple'
predicted_label = raw_predicted_label.replace('_', ' ').title()

# Encode gambar ke Base64 untuk ditampilkan di web
img_uploaded = base64.b64encode(img_bytes).decode("utf-8")

return render_template("index.html",
    img_uploaded=img_uploaded, predicted_label=predicted_label)
```

**Figure 6. Prediction Process Code**

## Application Interface

The developed web application is named "Know Your Fruits", featuring a simple and user-friendly interface. The application consists of four main pages Home, About, Predict, and Contact which can be accessed through the navigation menu at the top of the page.

The first page is the Home page, which serves as the initial view when users access the application. This page displays the title *"Know Your Fruits"* along with the tagline *"What and how is the state of your fruit"*, and includes a Get Started button that directs users to the classification feature. The Home page display is shown in Figure 7.

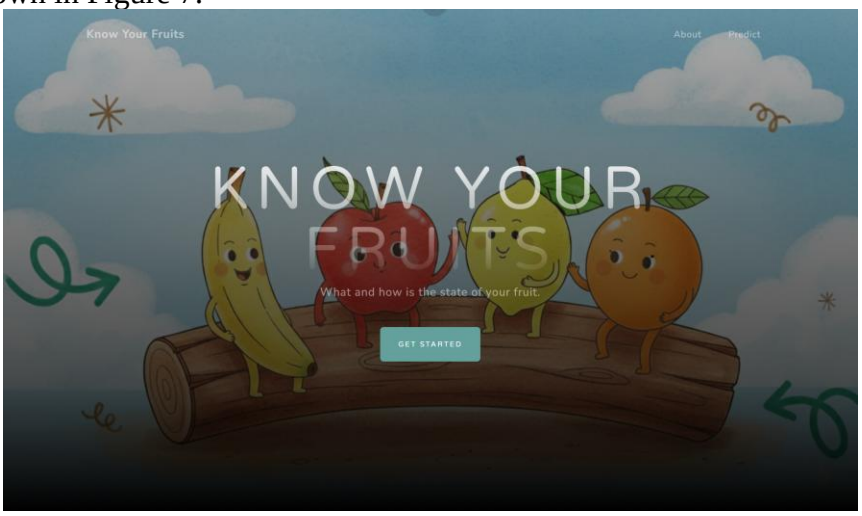


**Figure 7. Home Page View of the Know Your Fruits Application**

The second page is the About page, which provides a brief explanation of the Know Your Fruits application as a website that helps detect fruit types and their freshness conditions using artificial intelligence. The About page display is shown in Figure 8.

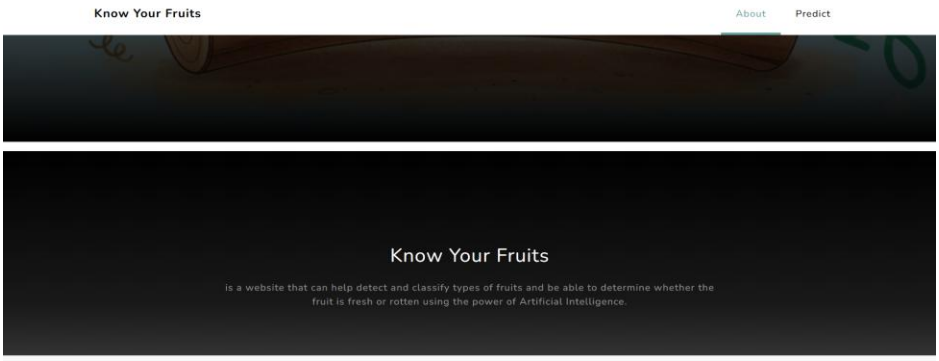


**Figure 8. About Page Display of the Know Your Fruits Application**

The third page is the Predict page, which is the main feature of the application. On this page, users can upload images of apples, bananas, or oranges using the available upload button, then click the Submit and Predict button to start the classification process. The Predict page display is shown in Figure 9.

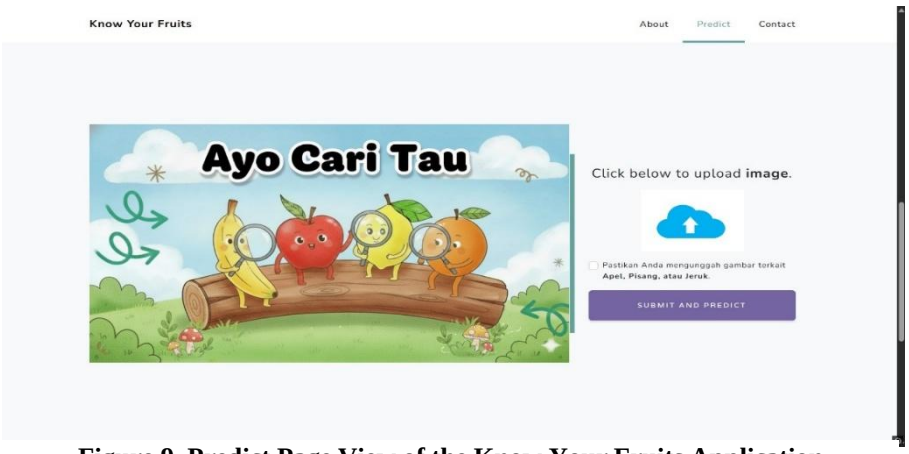


**Figure 9. Predict Page View of the Know Your Fruits Application**

After the user uploads an image and presses the Submit and Predict button, the system processes the image through a CNN model and displays the classification results. The predicted results are displayed as the fruit class name and its freshness level, for example, Fresh Apple, along with the uploaded image. The predicted results are shown in Figure 10 below.

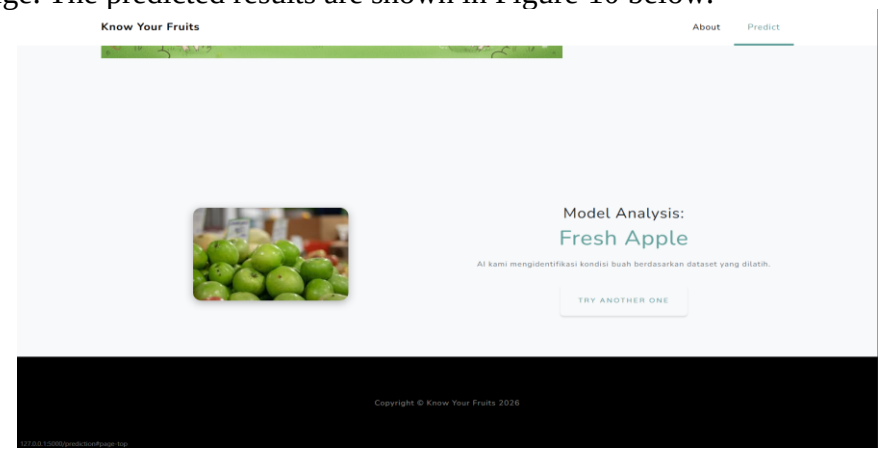


**Figure 10. Display of Know Your Fruits Application Prediction Results**

## Application Testing

Application testing is carried out using the Blackbox Testing method, which focuses on system functionality from the user's perspective without considering internal implementation details. This

method is chosen because it is suitable for verifying whether the system can produce correct outputs based on the given inputs.

The testing is conducted using 30 fruit images obtained from the internet with various file formats, including JPG, PNG, JFIF, and HEIF. These images are evenly distributed into 5 images for each of the six classes: fresh_apple, fresh_banana, fresh_orange, rotten_apple, rotten_orange, and rotten_banana. For each class, 3 images use a white background and 2 images use a varied (non-uniform) background. This distribution aims to evaluate the model's ability to recognize fruits under different background conditions.

Each image is tested directly through the Know Your Fruits web application by uploading the image on the Predict page and then clicking the Submit and Predict button. The uploaded image is automatically processed by resizing it to 150×150 pixels before being fed into the CNN model. The prediction results displayed by the application are then compared with the actual labels to determine whether the predictions are correct or incorrect, the complete testing results are presented in Table 8.

**Table 8. Black Box Results**

| No | Class | Sample | Background | Format File | Expectation | Prediction Results | Status |
|---|---|---|---|---|---|---|---|
| 1 | Fresh Apple | Sample 1 | White | JFIF | Fresh Apple | Fresh Apple | Correct |
| 2 | Fresh Apple | Sample 2 | White | JFIF | Fresh Apple | Fresh Apple | Correct |
| 3 | Fresh Apple | Sample 3 | White | JPG | Fresh Apple | Fresh Apple | Correct |
| 4 | Fresh Apple | Sample 4 | Free | JFIF | Fresh Apple | Fresh Apple | Correct |
| 5 | Fresh Apple | Sample 5 | Free | JPEG | Fresh Apple | Fresh Apple | Correct |
| 6 | Fresh Banana | Sample 6 | White | JPG | Fresh Banana | Fresh Banana | Correct |
| 7 | Fresh Banana | Sample 7 | White | JFIF | Fresh Banana | Fresh Banana | Correct |
| 8 | Fresh Banana | Sample 8 | White | JFIF | Fresh Banana | Fresh Banana | Correct |
| 9 | Fresh Banana | Sample 9 | Free | JPG | Fresh Banana | Fresh Banana | Correct |
| 10 | Fresh Banana | Sample 10 | Free | JPG | Fresh Banana | Fresh Banana | Correct |
| 11 | Fresh Orange | Sample 11 | White | JPEG | Fresh Orange | Fresh Orange | Correct |
| 12 | Fresh Orange | Sample 12 | White | JFIF | Fresh Orange | Fresh Orange | Correct |
| 13 | Fresh Orange | Sample 13 | White | JFIF | Fresh Orange | Fresh Orange | Correct |
| 14 | Fresh Orange | Sample 14 | Free | JPG | Fresh Orange | Fresh Orange | Correct |
| 15 | Fresh Orange | Sample 15 | Free | JPG | Fresh Orange | Fresh Orange | Correct |
| 16 | Rotten Apple | Sample 16 | White | JFIF | Rotten Apple | Rotten Apple | Correct |
| 17 | Rotten Apple | Sample 17 | White | JPEG | Rotten Apple | Rotten Apple | Correct |
| 18 | Rotten Apple | Sample 18 | White | JFIF | Rotten Apple | Rotten Apple | Correct |
| 19 | Rotten Apple | Sample 19 | Free | JPG | Rotten Apple | Rotten Apple | Correct |
| 20 | Rotten Apple | Sample 20 | Free | JPG | Rotten Apple | Rotten Banana | Wrong |
| 21 | Rotten Orange | Sample 21 | White | JPG | Rotten Orange | Rotten Orange | Correct |
| 22 | Rotten Orange | Sample 22 | White | JPG | Rotten Orange | Fresh Orange | Wrong |
| 23 | Rotten Orange | Sample 23 | White | JPG | Rotten Orange | Rotten Orange | Correct |
| 24 | Rotten Orange | Sample 24 | Free | HEIF | Rotten Orange | Rotten Orange | Correct |
| 25 | Rotten Orange | Sample 25 | Free | JPG | Rotten Orange | Rotten Orange | Correct |
| 26 | Rotten Banana | Sample 26 | White | JFIF | Rotten Banana | Rotten Banana | Correct |
| 27 | Rotten Banana | Sample 27 | White | JFIF | Rotten Banana | Rotten Banana | Correct |
| 28 | Rotten Banana | Sample 28 | White | JFIF | Rotten Banana | Rotten Banana | Correct |
| 29 | Rotten Banana | Sample 29 | Free | WEBP | Rotten Banana | Rotten Banana | Correct |
| 30 | Rotten Banana | Sample 30 | Free | WEBP | Rotten Banana | Rotten Banana | Correct |

Based on the results of functional testing using the Black Box Testing method on 30 independent image samples collected from the internet, the "Know Your Fruits" web application system generally demonstrates very good generalization performance and adaptability. The model is capable of recognizing fruit freshness characteristics across various shooting angles and different background conditions. Although the model shows a very high success rate, the testing results reveal certain limitations, with 2 image samples experiencing prediction failures (misclassification). The first case occurs when a rotten apple image is predicted as a rotten banana (Sample 20). The second case occurs when a rotten orange image is incorrectly identified as a fresh orange (Sample 22). Documentation of

these incorrect prediction results as displayed in the web application interface can be seen in Figures 11 and 12.

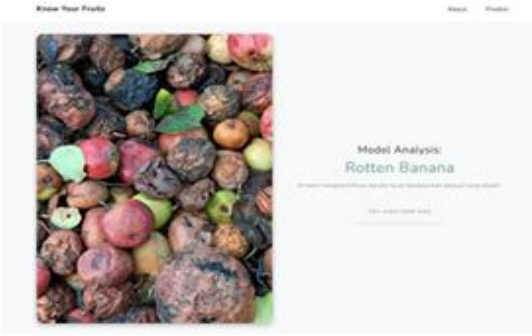


**Figure 11. Web Interface View**


**Figure 12. Web Interface View**

A closer examination of both misclassification cases reveals that the prediction failures were not caused by architectural weaknesses, but rather by external visual characteristics of the fruit images themselves. In the first case (Sample 20), a rotten apple was misclassified as a rotten banana due to severe geometric distortion caused by advanced decay, combined with a dominant dark brown color and wrinkled texture that closely resembles the visual appearance of a rotten banana. In the second case (Sample 22), a rotten orange was misclassified as a fresh orange because the fruit was still at an early stage of decay, where rotting signs were limited to small faint spots near the stem while the majority of the surface retained a bright dominant orange color, causing the model to assign higher probability to the fresh orange class. These findings are consistent with Kumar et al. (2025), who noted that degradation of standard visual features in decaying fruit frequently triggers feature extraction errors, leading the model to rely on color and texture overlap between classes rather than distinctive shape-based features.

## CONCLUSIONS

This study successfully demonstrated that a custom Sequential CNN architecture consisting of five progressive convolutional layers with increasing filter sizes (32, 64, 128, 256, and 512), combined with a dropout rate of 0.5 and EarlyStopping mechanism, is highly effective for multi-class fruit freshness classification. The training process demonstrated stable convergence without overfitting, as evidenced by the consistent alignment between training and validation accuracy throughout all epochs. The model achieved a Test Accuracy of 98.92% and a Test Loss of 0.1404 on 840 testing images, with the rotten_orange class exhibiting the highest misclassification rate among all six classes, indicating that visually ambiguous decay characteristics remain the most challenging aspect for the model. Furthermore, the trained model was successfully deployed into a functional web-based application named "Know Your Fruits", demonstrating that a deep learning model trained in a cloud environment can be effectively transferred into a real-world production system while maintaining classification

performance consistency. Based on Black Box Testing using 30 independent real-world image samples, the system achieved a functional accuracy of 93.33%, with the remaining 6.67% misclassification attributed to external visual factors, specifically severe geometric distortion caused by advanced decay and early-stage decay conditions where rotting signs remain visually indistinct, as supported by Kumar et al. (2025).

## REFERENCES

Arkadia, A., Ayu Damayanti, S., & Sandya Prasvita, D. (2021). Klasifikasi Buah Mangga Badami Untuk Menentukan Tingkat Kematangan dengan Metode CNN. *Prosiding Seminar Nasional Mahasiswa Ilmu Komputer dan Aplikasinya (SENAMIKA) Jakarta-Indonesia*, 2(2), 158–165. https://conference.upnvj.ac.id/index.php/senamika/article/view/1813

Kumar, S., Sharma, A., Singh, R., & Patel, M. (2025). Fresh or Rotten? Enhancing Rotten Fruit Detection with Deep Learning and Gaussian Filtering. *IEEE Transactions on Agri-Food Electronics*, 3(1), 45–58. https://doi.org/10.1109/TAFE.2025.3412345

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. *Nature*, 521(7553), 436–444. https://doi.org/10.1038/nature14539

Mikasari, W., Hamzah, A., & Nurjanah, S. (2021). Perubahan Kualitas Fisik Buah Jeruk Selama Penyimpanan. *Jurnal Teknologi Pertanian*, 12(2), 85–92.

Putri, M. S., & Kurniawan, A. (2023). Pendeteksian Kesegaran Buah Jeruk Berdasarkan Citra Permukaan dengan Convolutional Neural Network. *Prosiding Seminar Nasional Teknologi dan Informatika (SENTIKA)*, 78–85.

Russell, S. J., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

Sari, A. N., & Pratama, R. H. (2022). Klasifikasi Kesegaran Buah Apel Menggunakan Convolutional Neural Network. *Jurnal Teknologi dan Rekayasa*, 8(2), 45–53.

Swoyam. (2023). *Fresh and Stale Classification* [Data set]. Kaggle. https://www.kaggle.com/datasets/swoyam2609/fresh-and-stale-classification