MAKERERE UNIVERSITY

COLLEGE OF COMPUTING AND INFORMATION SCIENCES

DEPARTMENT OF NETWORKS

COURSE **BACHELOR OF SCIENCE IN SOFTWARE ENGINEERING**

COURSE NAME **SOFTWARE ENGINEERING**
MINI PROJECT 2

COURSE CODE **BSE2301**

GROUP FIVE **TOMATO DISEASE MACHINE LEARNING**

NAME	REG NUMBER	STUD NO
KATURAMU EDGAR	22/U/21756/PS	2200721756
TUKWASIIBWE MARTIN	22/U/21816/EVE	2200721816
KANYESIGYE WILLIAM	22/U/22576	2200721799
AINE LEVI	22/U/2903/EVE	2200702903
NIWAMANYA JAMES	22/U/21791/EVE	

GROUP 5 TOMATO DISEASE MACHINE LEARNING PROJECT REPORT

1. INTRODUCTION

This report documents the process and results of Group 5's project, which aimed to develop a machine-learning model to classify images of tomatoes into three categories: reject, ripe, and unripe.

Our dataset contained 2400 images of tomatoes, evenly distributed across three classes:
Reject: Images of tomatoes with abnormalities or defects.
Ripe: Images of healthy, fully ripened tomatoes.
Unripe: Images of tomatoes that are not yet fully ripe.
The dataset was organized into three sets:
Training set: Used to train the machine learning model.
Validation set: Used to fine-tune the model's hyperparameters.
Test set: Used to evaluate the final model's performance on unseen data.

2. METHODOLOGY

2.1 Importing Libraries

To build and evaluate the model, several key libraries were utilized:

- TensorFlow for building and training the neural network.
- Matplotlib and Seaborn for data visualization and analysis.
- Pandas for data manipulation and preprocessing.

2.2 Data Processing

Data preprocessing was carried out using Keras, a high-level API of TensorFlow. Image data was loaded into the training_set and validation_set using Keras' ImageDataGenerator, which also applied data augmentation techniques to improve model generalization.

2.3 Model Building with Convolutional Neural Networks (CNN)

A CNN was constructed using TensorFlow's Keras API. The architecture included various layers:

InputLayer: Defines the input shape of the images (128x128 pixels with 3 color channels.

Conv2D: Convolutional layers for feature extraction.

MaxPool2D: Pooling layers to reduce the dimensionality.

Flatten: A layer to flatten the feature map into a one-dimensional vector.

Dense: Fully connected layers for classification.

Dropout: A regularization technique to prevent overfitting.

```
model = Sequential()
✓ 0.0s                                                                    Python


# Add an InputLayer to define the input shape
model.add(InputLayer(input_shape=(128, 128, 3)))

# Add Conv2D and other layers
model.add(Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'))
model.add(Conv2D(filters=64, kernel_size=3, activation='relu'))
model.add(                                    strides=2))
          (import) Conv2D: Any
model.add(Conv2D(filters=128, kernel_size=3, padding='same', activation='relu'))
model.add(Conv2D(filters=128, kernel_size=3, activation='relu'))
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Conv2D(filters=256, kernel_size=3, padding='same', activation='relu'))
model.add(Conv2D(filters=256, kernel_size=3, activation='relu'))
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Conv2D(filters=512, kernel_size=3, padding='same', activation='relu'))
model.add(Conv2D(filters=512, kernel_size=3, activation='relu'))
model.add(MaxPool2D(pool_size=2, strides=2))

model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(units=128, activation='relu'))

model.add(Dropout(0.4))

model.add(Dense(units=3, activation='softmax'))  # Ensure the output layer has 3 units for your classes
```

2.4 Model Compilation

```
model.compile(optimizer = tf.keras.optimizers.Adam(learning_rate = 0.0001), loss =
'categorical_crossentropy', metrics = ['accuracy'])
```

The model was compiled with the Adam optimizer, a learning rate of 0.0001, and the categorical crossentropy loss function. The metric for evaluating the model's performance was accuracy.

2.5 Model Training

The model was trained using the training set for 30 epochs. The training process involved adjusting the model's parameters to minimize the loss function and improve accuracy. The output below shows the model's performance over 30 epochs, with each epoch representing one complete pass through the training dataset. Key metrics like loss and accuracy are reported for both the training and validation sets. The model improved over time, with decreasing loss and increasing accuracy on both sets, indicating successful training.

Model training

```
training_history = model.fit(x = training_set, validation_data = validation_set, epochs = 30)
```

[66] ✓ 113m 44.2s

```
Epoch 1/30
75/75 [==============================] - 233s 3s/step - loss: 0.1859 - accuracy: 0.9229 - val_loss: 0.1426 - val_accuracy: 0.9396
Epoch 2/30
75/75 [==============================] - 233s 3s/step - loss: 0.1604 - accuracy: 0.9304 - val_loss: 0.1145 - val_accuracy: 0.9542
Epoch 3/30
75/75 [==============================] - 231s 3s/step - loss: 0.1577 - accuracy: 0.9342 - val_loss: 0.1367 - val_accuracy: 0.9454
Epoch 4/30
75/75 [==============================] - 231s 3s/step - loss: 0.1760 - accuracy: 0.9212 - val_loss: 0.1114 - val_accuracy: 0.9583
Epoch 5/30
75/75 [==============================] - 235s 3s/step - loss: 0.1331 - accuracy: 0.9458 - val_loss: 0.1279 - val_accuracy: 0.9421
Epoch 6/30
75/75 [==============================] - 230s 3s/step - loss: 0.1195 - accuracy: 0.9508 - val_loss: 0.1150 - val_accuracy: 0.9471
Epoch 7/30
75/75 [==============================] - 230s 3s/step - loss: 0.0982 - accuracy: 0.9600 - val_loss: 0.0858 - val_accuracy: 0.9646
Epoch 8/30
75/75 [==============================] - 228s 3s/step - loss: 0.1214 - accuracy: 0.9533 - val_loss: 0.0721 - val_accuracy: 0.9783
Epoch 9/30
75/75 [==============================] - 237s 3s/step - loss: 0.0887 - accuracy: 0.9638 - val_loss: 0.1170 - val_accuracy: 0.9529
Epoch 10/30
75/75 [==============================] - 224s 3s/step - loss: 0.0885 - accuracy: 0.9671 - val_loss: 0.0373 - val_accuracy: 0.9887
Epoch 11/30
75/75 [==============================] - 224s 3s/step - loss: 0.0696 - accuracy: 0.9750 - val_loss: 0.0708 - val_accuracy: 0.9700
Epoch 12/30
75/75 [==============================] - 224s 3s/step - loss: 0.0621 - accuracy: 0.9767 - val_loss: 0.0320 - val_accuracy: 0.9887
Epoch 13/30
...
Epoch 29/30
75/75 [==============================] - 229s 3s/step - loss: 0.0052 - accuracy: 0.9979 - val_loss: 0.0025 - val_accuracy: 0.9987
```

3. MODEL EVALUATION

3.1 Training Set Evaluation

The model achieved impressive accuracy on the training set, indicating a strong capability to learn and classify the data.

Training Loss: 0.0038

Training Accuracy: 99.83%

3.2 Validation Set Evaluation

Similar results were observed on the validation set, demonstrating the model's ability to generalize to unseen data.

Validation Loss: 0.0038

Validation Accuracy: 99.83%

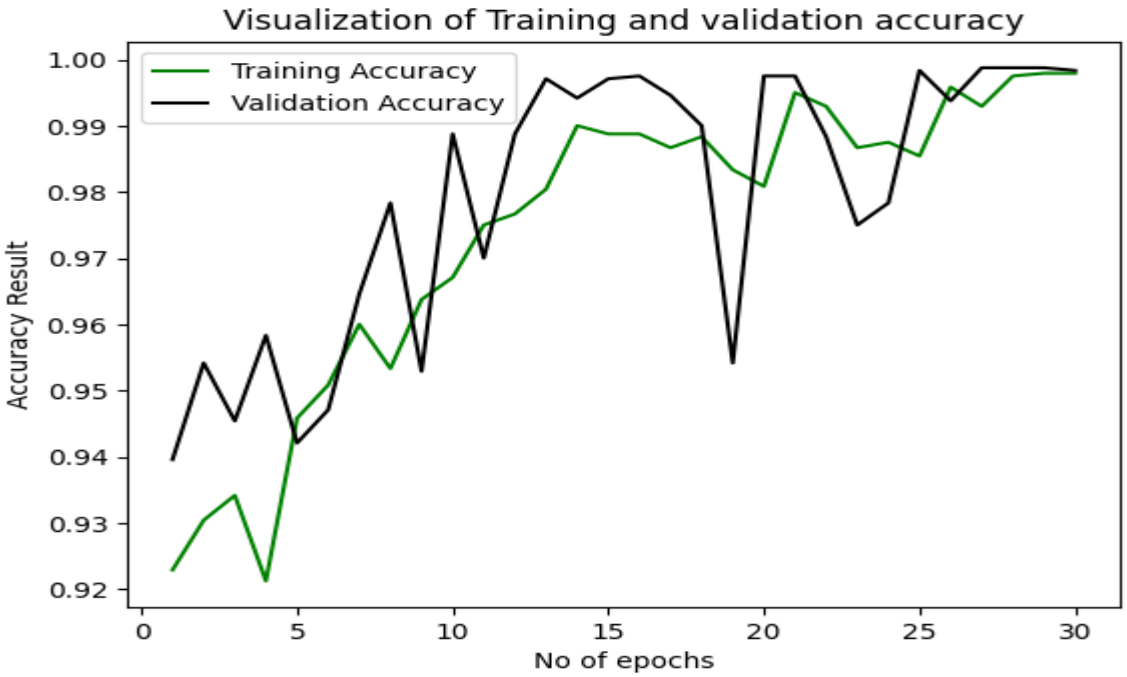
3.3 Performance Metrics

The model's performance was evaluated using precision, recall, and F1-score, calculated for each class. The results were perfect across all metrics, reflecting the model's high accuracy in distinguishing between the three classes. The results below indicate perfect performance across all classes (Reject, Ripe, and Unripe), with 100% accuracy, precision, recall, and F1-score, suggesting an exceptionally well-performing model.

```
from sklearn.metrics import classification_report, confusion_matrix
✓ 0.0s

print(classification_report(Y_true, predicted_categories, target_names = class_name))
✓ 0.0s
              precision    recall  f1-score   support

      Reject       1.00      1.00      1.00       800
        Ripe       1.00      1.00      1.00       800
      Unripe       1.00      1.00      1.00       800

    accuracy                           1.00      2400
   macro avg       1.00      1.00      1.00      2400
weighted avg       1.00      1.00      1.00      2400
```

3.4 Visualizations to uncover patterns and insights in the data

Visualization of Training and Validation accuracy

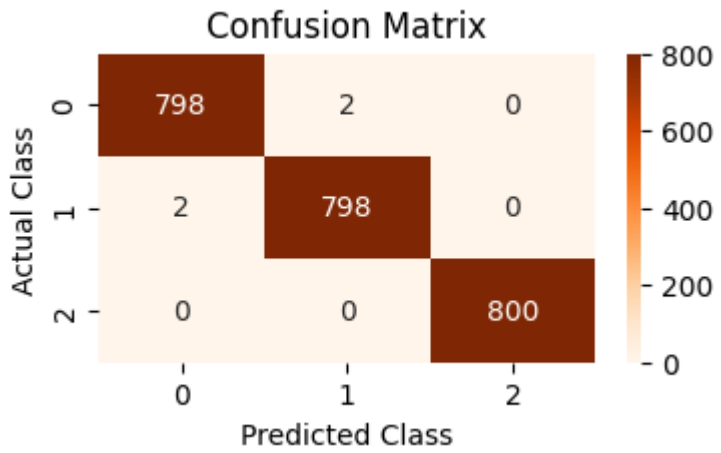


The image above shows the training and validation accuracy of a machine learning model over 30 epochs. The training accuracy starts at around 92% and quickly increases to nearly 100% within the first few epochs. However, the validation accuracy, which measures the model's performance shows a more fluctuating pattern over time but later meets at the same point with the training accuracy.

Visualization of Training and Validation Loss

The image above shows the training and validation loss of a machine learning model over 30 epochs. Both the training and validation loss decrease significantly in the initial epochs, indicating that the model is learning effectively. The training loss continues to decrease steadily, reaching a value close to 0 by the end of the training period. The validation loss also decreases initially but plateaus around 0.025 after approximately 15 epochs. This suggests that while the model continues to improve on the training data, its performance

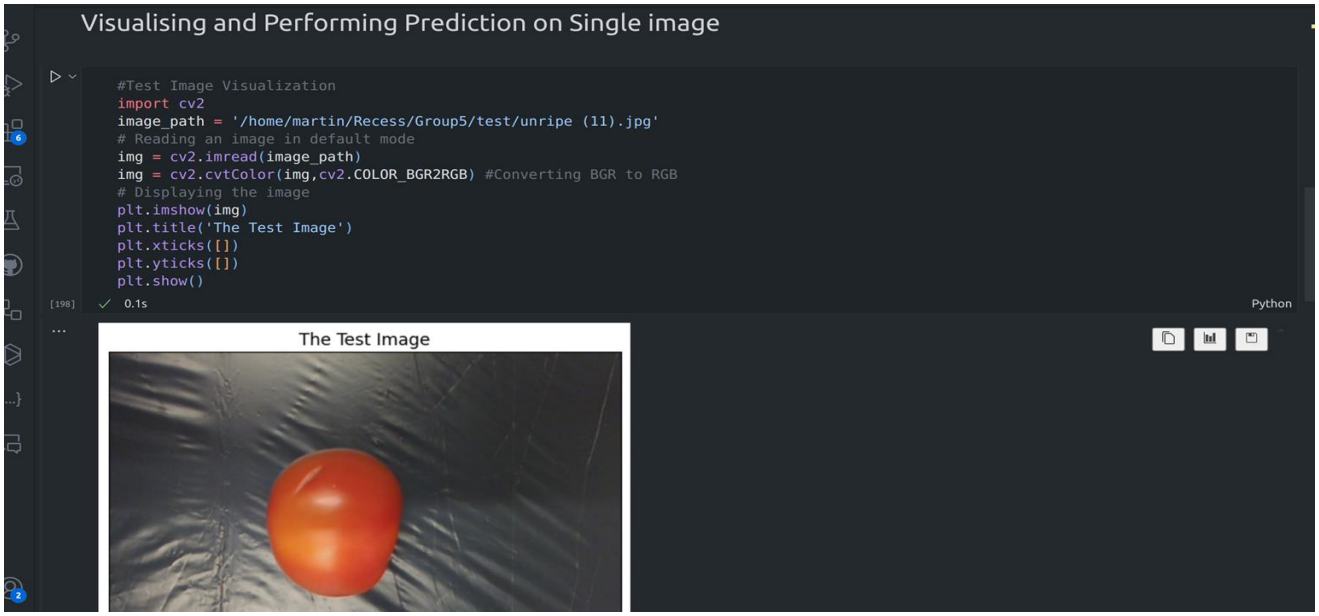
Confusion Matrix



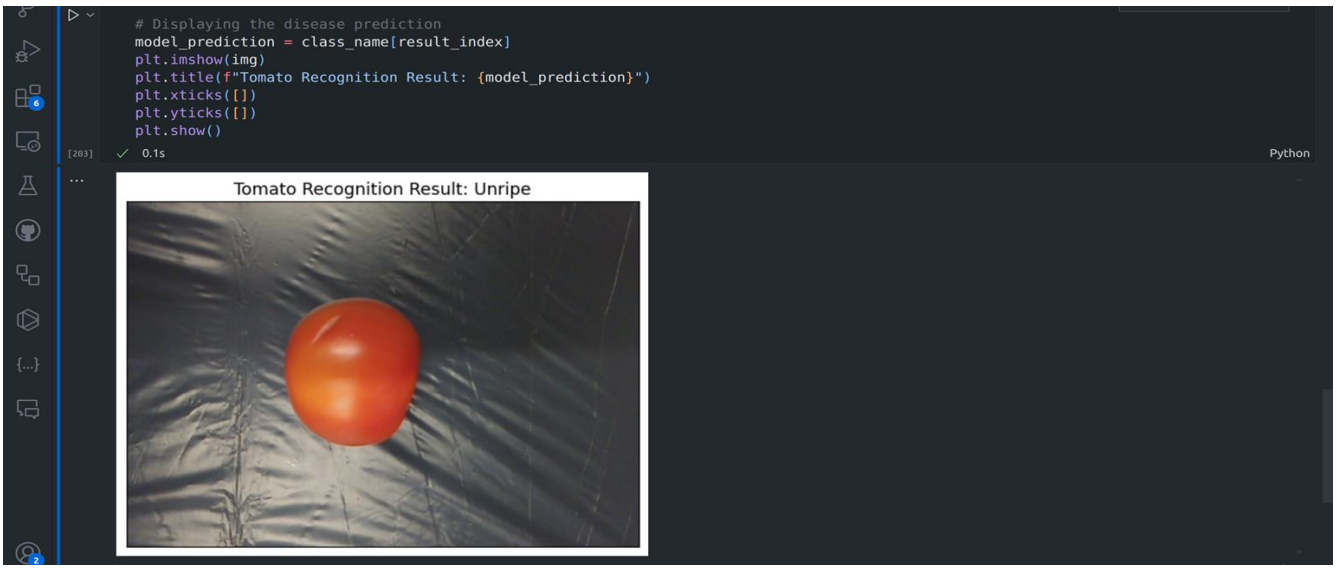
The confusion matrix provides insights into the model's performance in classifying tomatoes into the categories 'Reject', 'Ripe', and 'Unripe'. The matrix reveals an almost perfect classification accuracy. There are only two instances of misclassification, both occurring between the 'Reject' and 'Ripe' categories. This indicates that the model is exceptionally proficient in distinguishing 'Unripe' tomatoes from the other two classes. However, there's still a slight confusion between 'Reject' and 'Ripe' tomatoes, which could be an area for further model improvement if higher precision is required for these categories. Overall, the model exhibits outstanding performance in this classification task.

3.5 Testing our trained model with sample images for prediction

The code in the illustration below shows the image of a tomato from the specified file path, converts the color space from BGR to RGB, and displays the image with a title The Test Image below.



The image below shows how the model has predicted the tomato to be "Unripe". The code also includes visualization steps to display the input image with the corresponding prediction as a title.



4. CONCLUSION

The project successfully developed a CNN model that excels in classifying tomato images into reject, ripe, and unripe categories. The model's outstanding performance on both the training and validation datasets, coupled with perfect precision, recall, and F1-scores, underscores its potential for practical applications in agriculture and food industry settings. Future work could focus on further improving the model's precision between the 'reject' and 'ripe' categories and exploring its applicability to other crops and diseases.