Classification of Guava Varieties based on features of the leaves

Team Name: CodePals

Table of Contents

1. GitHub Link:

https://github.com/Mvikum/multiclass_guava_classifier-.git

2. Introduction

The objective of this assignment is to develop a multiclass classifier to accurately distinguish between three guava species: Indonesian guava, green guava, and white guava. Accurate classification of tree species is important for various applications, including biodiversity studies and also my colleagues of biological department will find this helpful. Visual identification of these guava species can be challenging due to their similar morphological characteristics. Therefore, a machine learning approach is employed to enhance the accuracy and reliability of species classification.

- The primary objectives of this project are:
  - To measure and record specific features of leaves from different guava species.
  - To analyze the data and classify the trees based on these features.
  - To determine the accuracy of our classification method.

3. Overview of Selected Species

The three guava species selected for this study are commonly found in the university garden. But when data collecting we gathered data from outside too. Each species has unique botanical traits that are not easily distinguishable by the human eye. Indonesian guava, green guava, and white guava share similar leaf shapes and sizes, making visual identification prone to errors. The use of quantitative features derived from the leaves, such as area, length-to-width ratio, petiole length, number of vein pairs, and vein angle ratio, provides a more objective basis for classification.

3.1. White Guava (Psidium guajava)

3.1.1. Description:

- Fruit: The fruit of the White Guava is typically pale green or yellowish on the outside with white flesh inside. It is known for its sweet and mildly tart flavor.

- Leaves: The leaves are elliptical, smooth, and dark green. They have a slightly leathery texture with pronounced veins.

- Flowers: White Guava flowers are small, white, and fragrant, with five petals and numerous stamens.

3.1.2. Visual Challenges:

- The external appearance of the fruit can be similar to other guava species, making it hard to identify without cutting open the fruit to see the flesh.

- During the non-fruiting season, the trees of White Guava look almost identical to other guava species, especially in terms of leaf shape and texture.

3.2. **Green Guava (Psidium guajava var. Green)**

3.2.1. Description:

- Fruit: Green Guava produces green-skinned fruit with light green flesh. The taste is generally sweet with a hint of acidity.
- Leaves: Similar to the White Guava, the leaves are dark green, elliptical, and have a smooth texture. The veins are prominent.
- Flowers: The flowers are white, with five petals, and are highly fragrant, attracting various pollinators.

3.2.2. Visual Challenges:

- The leaves and flowers of Green Guava are very similar to those of White and Red Guava, making it difficult to distinguish the species based on these features alone.
- The fruit’s external color is not always a reliable indicator, especially if the fruit is not fully ripe.

3.3. **Indonesian Guava (Psidium guajava var. Red)**

3.3.1. Description:

- Fruit: The fruit of Indonesian Guava has a green to yellowish exterior with pink to red flesh. It is often sweeter and more aromatic compared to other guava species.
- Leaves: The leaves are also dark green, elliptical, and exhibit a similar texture and vein pattern to the other guava species.
- Flowers: Indonesian Guava flowers are white, with the typical structure of five petals and numerous stamens.

3.3.2. Visual Challenges:

- The Indonesian Guava’s leaves and flowers are nearly indistinguishable from those of White and Green Guava.
- The distinct red flesh of the fruit is only visible when the fruit is cut open, which is not possible when identifying trees without fruit.

3.4. **Discussion**

- **Challenges in Visual Identification:** The primary challenge in distinguishing these guava species lies in their remarkable similarity in leaf and flower morphology. The leaves of White, Green, and Indonesian Guava are all elliptical, dark green, and have pronounced veins, making it extremely difficult to tell them apart based on foliage alone. Their flowers, which are all small, white, and fragrant, add to the complexity of visual differentiation.

- **Reliance on Fruit for Identification:** During the fruiting season, the identification becomes slightly easier as the internal color of the fruit can be observed. White Guava has white flesh, Green Guava has light green flesh, and Indonesian Guava has pink to red flesh. However, this method is only useful when fruits are available, which is typically seasonal.
- **Use of Numerical Data:** Given the visual similarities, numerical data becomes crucial for accurate identification. Specific measurements and characteristics such as leaf area, petiole length, number of vein pairs, and the ratio of vein angles can provide the necessary distinctions between these species. These quantitative features, which are often subtle and require precise measurement, enable us to differentiate between the species with higher accuracy.

By focusing on numerical data and developing a robust classification model, we can overcome the limitations of visual identification and ensure accurate species recognition, even outside the fruiting season.

4. Dataset Preparation

We have met Dr. Asanga Wijetunga to select what are the features of the leaves we are considering to prepare the dataset. Initially we gather feature as thickness, length and width of the leaf, area of the leaf, angle between veins. Since leaves are very thin, we can’t measure it, without using special equipment. So, we selected following features for populate the dataset.

1. Area (cm^2).
2. Ratio between length and width.
3. Petiole length (mm).
4. Number of veins pairs.
5. Ratio between angle of vein.

4.1. Methods of measuring numerical data values of leaves

4.1.1. Area

- **Materials Needed**
  - We used a graph sheet, a pencil, and the leaf we wanted to measure.
- **Measuring the Area of a Leaf Using a Graph Sheet**
  - To measure the area of a leaf, we followed these steps:

1. **Placing the Leaf:**
   - We placed the leaf flat on the graph sheet to ensure it was properly positioned for tracing.
2. **Tracing the Leaf:**
   - We carefully traced around the edge of the leaf with a pencil, making sure to follow the outline accurately.

3. **Counting the Squares:**
   - **Whole Squares:**
     - We counted all the full squares within the traced outline. These are the squares that were completely inside the leaf's outline.

   - **Partial Squares:**
     - We also counted the partial squares that the outline crossed. We estimated how many of these partial squares would add up to whole squares. For instance, if there were approximately 20 half-squares, we considered this as 10 full squares.

   - **Total Squares:**
     - We added the number of whole squares to the equivalent number of whole squares from the partial ones. For

example, if we counted 50 whole squares and estimated 10 more whole squares from the partial ones, the total came to 60 squares.

4. **Calculating the Area:**
    - Each square on the graph sheet represents a specific area (e.g., 1 square centimeter). By multiplying the total number of squares by the area each square represents, we calculated the total area of the leaf. In this case, if each square is 1 square centimeter, the total area of the leaf is 60 square centimeters.

- Thus, we determined the area of the leaf to be 60 square centimeters using this method.

4.1.2. Ratio between length and width.

- To measure the ratio between the length and width of a leaf, we followed these steps:
    1. **Measuring the Length**:
        a. We placed the ruler along the longest part of the leaf.
        b. We recorded the measurement in centimeters. This is the length of the leaf.
    2. **Measuring the Width**:
        a. We placed the ruler across the widest part of the leaf.
        b. We recorded the measurement in centimeters. This is the width of the leaf.
    3. **Calculating the Ratio**:
        a. We divided the length by the width to find the ratio.
        b. We wrote down the final value as a decimal.

- For example, if the length of the leaf is 12 cm and the width is 4 cm:
    a. We divided 12 by 4, which equals 3.
    b. The ratio between the length and width of the leaf is 3.0.

- Thus, we determined the ratio by dividing the length by the width and writing the answer as a decimal.

4.1.3. Petiole length (mm).

- **Materials Needed:** We used a ruler and the leaf we wanted to measure.
- To measure the petiole length of a leaf, we followed these steps:

1. **Identifying the Petiole**:
    a. We located the petiole, which is the stalk that attaches the leaf to the stem.
2. **Placing the Ruler**:
    a. We placed the ruler at the base of the petiole, where it connects to the stem.
    b. We extended the ruler along the length of the petiole to its tip, where it connects to the leaf blade.
3. **Reading the Measurement**:
    a. We carefully read the measurement on the ruler at the tip of the petiole.
    b. We recorded the length in millimeters (mm).

- By following these steps, we measured the petiole length of the leaf accurately using a ruler.

4.1.4. Number of veins pairs.

- **Materials Needed**: We used the leaf we wanted to examine.
- To count the number of vein pairs in a leaf, we followed these steps:

1. **Identifying the Veins**:
    a. We looked at the leaf closely to identify the veins.
    b. We focused only on the dark veins, as these are more visible.
2. **Counting the Vein Pairs**:
    a. We started from the base of the leaf and moved towards the tip.
    b. We counted each pair of veins, making sure to count only the dark ones.
    c. We recorded the total number of vein pairs we counted.

By following these steps, we accurately counted the number of dark vein pairs in the leaf.

4.1.5. Ratio between angles of veins

- **Materials Needed**: We used a protractor and the leaf we wanted to measure.
- To measure the ratio between the angles of veins in a leaf, we followed these steps:

1. **Identifying the Veins**:
    a. We located the middle vein (main vein) and the side veins of the leaf.
    b. We found the widest part of the leaf where we would measure the angles.
2. **Measuring the Angles**:
    a. We placed the protractor at the point where a side vein meets the middle vein.

   b. We measured the angle formed by the side vein and the middle vein on one side of the leaf.
   c. We recorded this angle.
   d. We repeated the measurement for the side vein on the opposite side of the leaf and recorded this angle.

3. **Calculating the Mean Angle**:
   a. We added the two measured angles together.
   b. We divided the sum by 2 to find the mean angle.

4. **Finding the Ratio**:
   a. We determined the ratio of the mean angle by comparing it to a reference angle or another measured angle as needed.

4.2. **Detailed Calculations**

4.2.1. Leaf Area Calculation

- To calculate the leaf area using a graph sheet:
  - Place the leaf on the graph sheet.
  - Trace the outline of the leaf.
  - Count the number of full squares within the traced outline.
  - Estimate the number of partial squares that add up to whole squares.
  - Multiply the total number of squares by the area each square represents.
- Example:
  - Full squares: 50
  - Partial squares: Equivalent to 10 full squares
  - Total squares: 60
  - Each square represents 1 cm^2
  - Leaf area = 60 cm^2

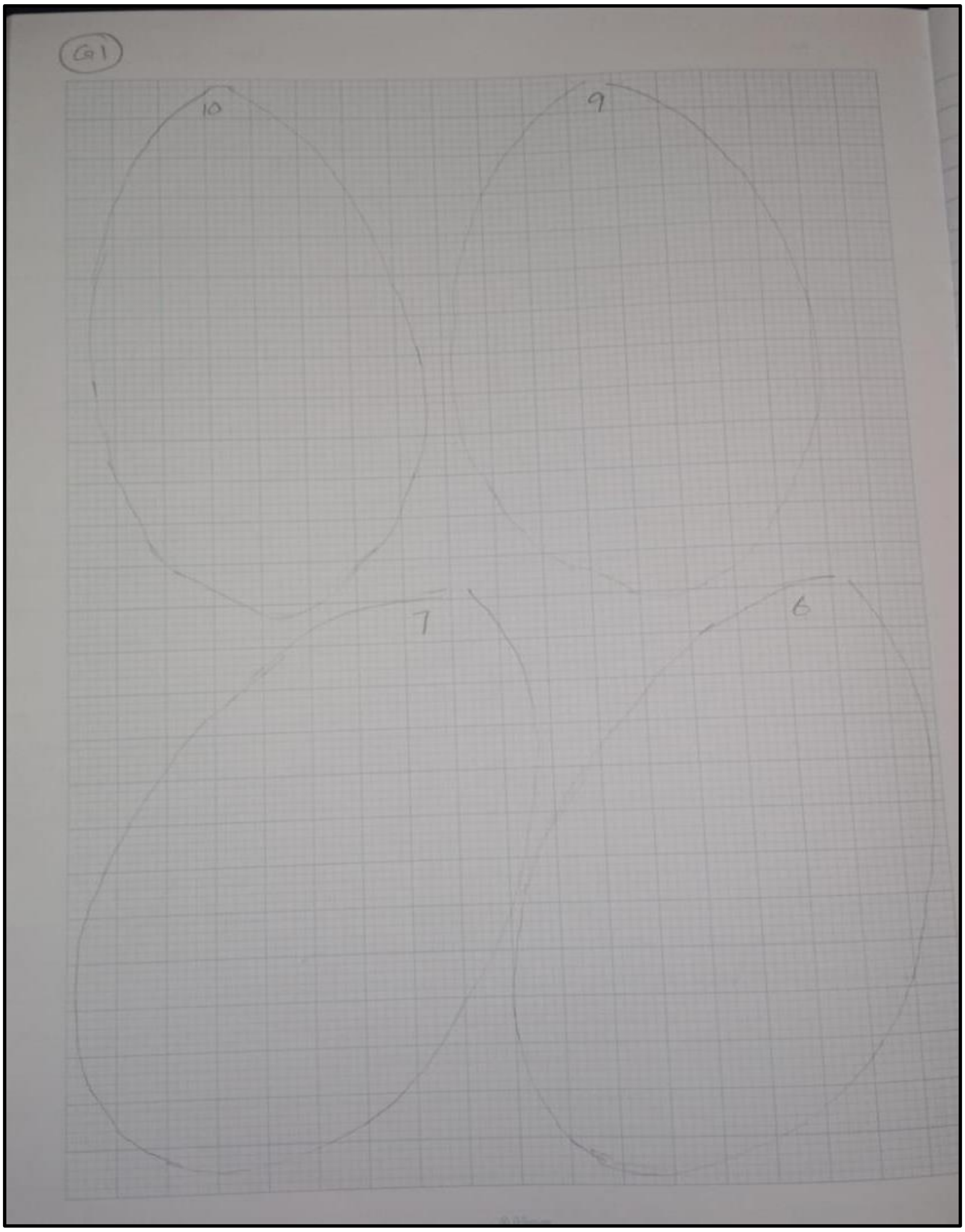


Figure 1: Calculating Area of the leaf

4.2.2. Ratio Between Length and Width

- To calculate the ratio between length and width of the leaf:
  - Measure the length of the leaf.
  - Measure the width of the leaf.
  - Divide the length by the width.
- Example:
  - Length = 12 cm
  - Width = 4 cm
  - Ratio = Length / Width = 12 / 4 = 3.0

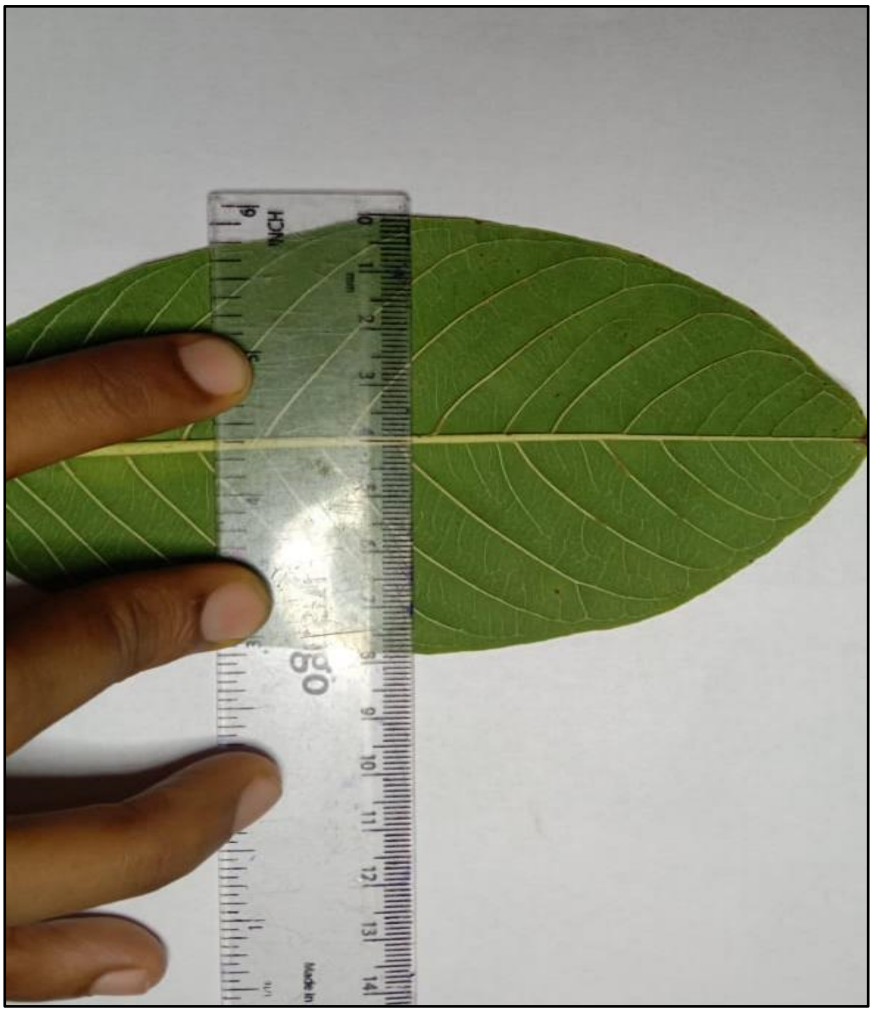


Figure 2: Calculating the width of the leave

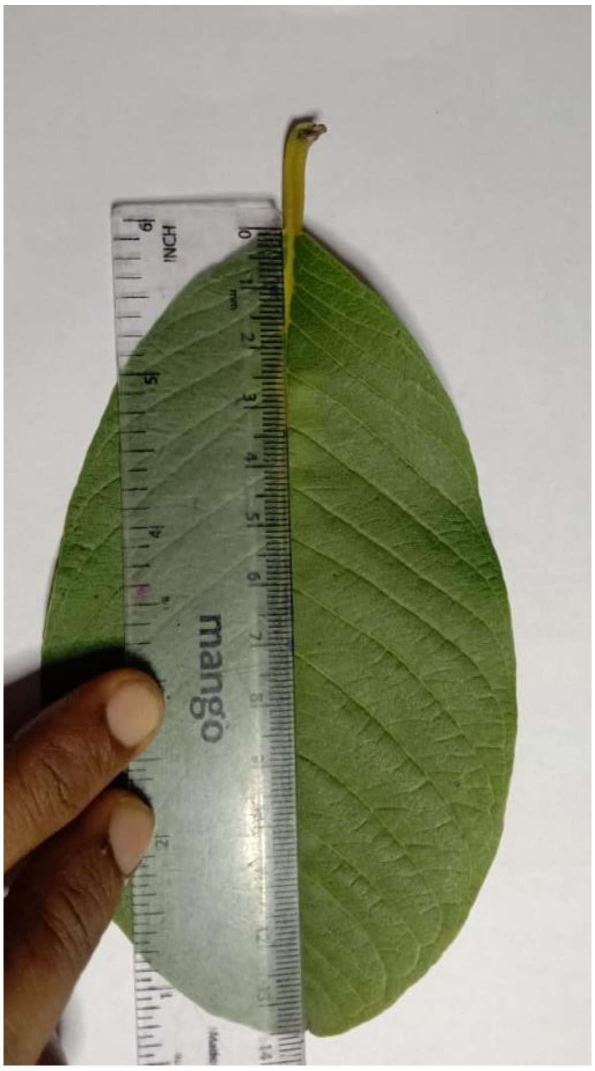


Figure 3:Measure the length of the leave

4.2.3. Petiole Length Measurement

- To measure the petiole length:
  - Place the ruler at the base of the petiole.
  - Measure to the tip where the petiole meets the leaf blade.
  - Record the length in millimeters (mm).
- Example:
  - Petiole length = 15 mm

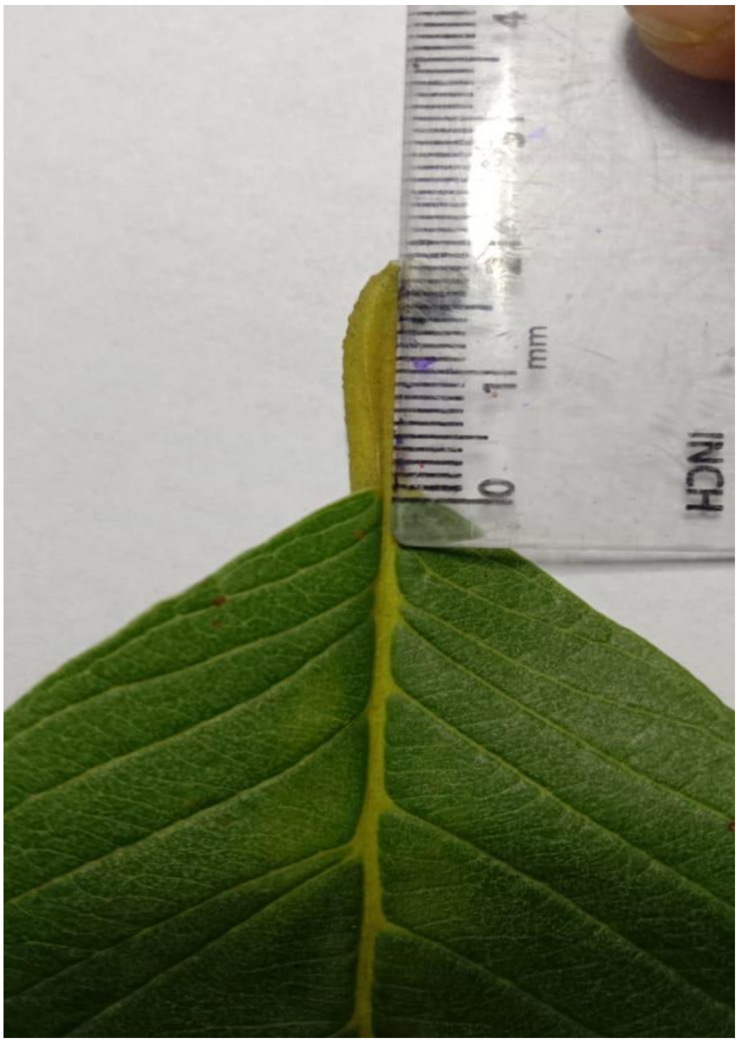


Figure 4: Measure the Petiole Length

4.2.4. Number of Vein Pairs

- To count the number of vein pairs:
  - Identify the veins starting from the base of the leaf.
  - Count each pair of veins up to the tip.
  - Record the total number of vein pairs.
- Example:
  - Number of vein pairs = 7

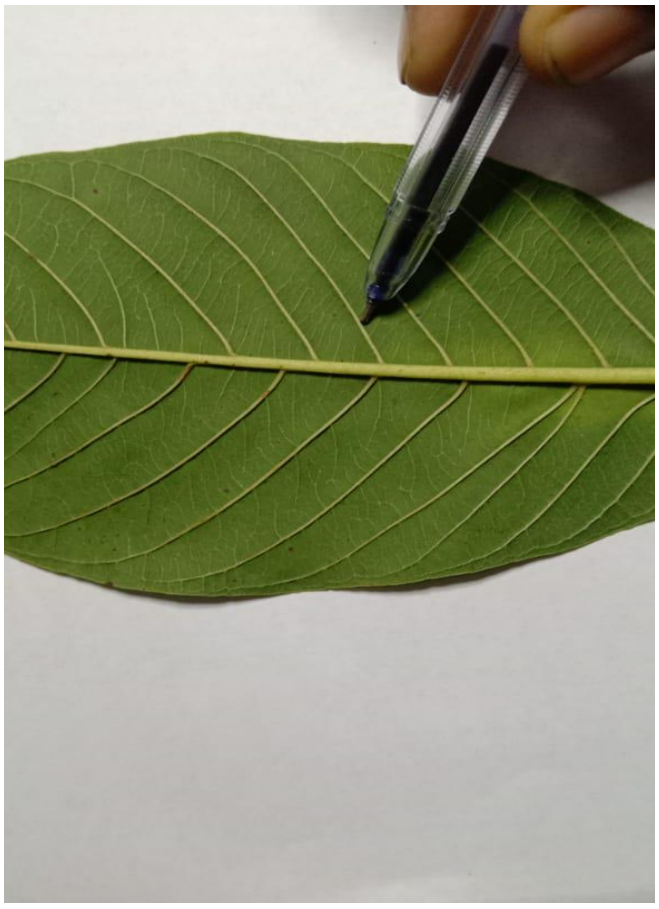


Figure 5: Counting number of Vein Pairs

4.2.5. Vein Angle Measurement

- To measure the vein angles:
    - Place the protractor at the point where a side vein meets the middle vein.
    - Measure the angle on both sides.
    - Calculate the mean angle.
- Example:
    - Angle 1 = 32°
    - Angle 2 = 28°
    - Mean angle = (32 + 28) / 2 = 30°

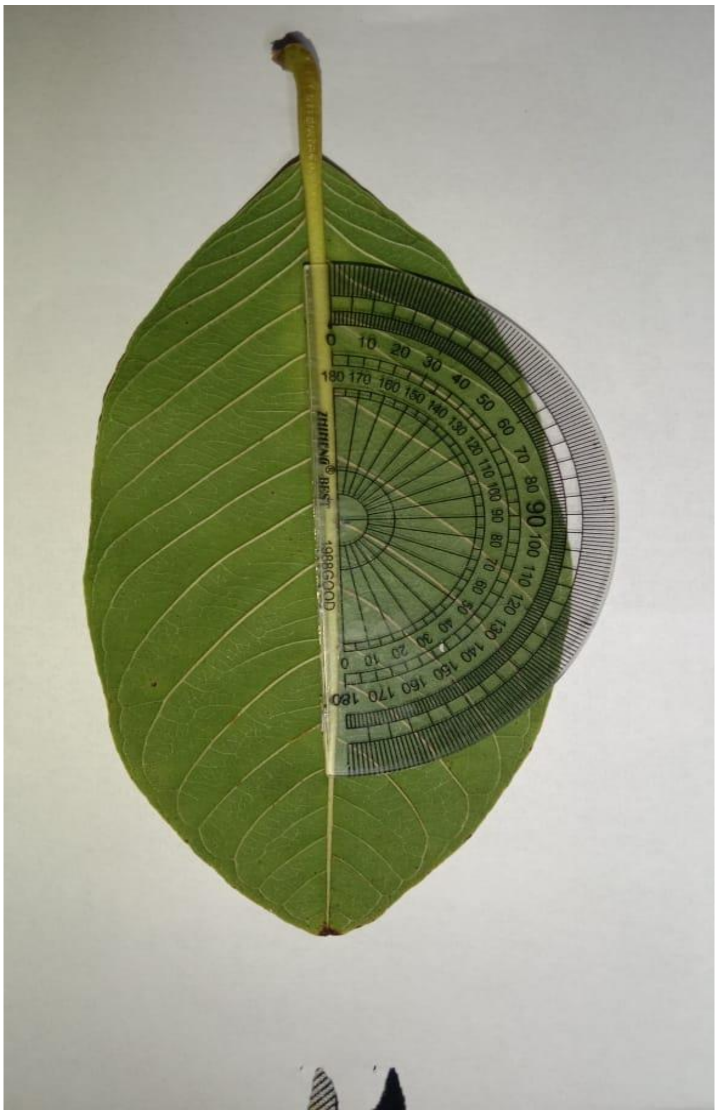


Figure 6:Measure the Angle

5. Data Analysis

- In this section, we explain how we used Python and AI methods to analyze the data and develop our tree classifier.

5.1. Data Preprocessing:

5.1.1. Loading the Data:

- We loaded the dataset using pandas from a CSV file named data.csv with a semicolon (;) delimiter.
- We displayed the first few rows and the information of the dataset to ensure it was loaded correctly.
- The target variable was identified as the column label, and the feature variables were all other columns except label.

```
#python
import pandas as pd

# Load the dataset
df = pd.read_csv('data.csv', delimiter=';')

# Display the first few rows to ensure it's loaded correctly
print(df.head())
print(df.info())


# Identify the target variable (assuming the column name is 'label')

target = df['label']
```

Figure 7: Code segment for loading dataset

5.1.2. Splitting the Data:

- We split the data into training and testing sets using train_test_split from sklearn. We used 80% of the data for training and 20% for testing, with a **random state of 42 to ensure reproducibility**.

```
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
```

Figure 8: Code segment for splitting data

5.1.3. Encoding the Target Variable:

- We used OneHotEncoder from sklearn to convert the target variable into a one-hot encoded format, as our model requires categorical data in this format.

```
#python
from sklearn.preprocessing import OneHotEncoder

# Initialize the OneHotEncoder
encoder = OneHotEncoder(sparse=False)

# Fit and transform the training labels
y_train_encoded = encoder.fit_transform(y_train.values.reshape(-1, 1))

# Transform the testing labels
y_test_encoded = encoder.transform(y_test.values.reshape(-1, 1))
```

Figure 9

5.1.4. Building the Neural Network Model:

- We used keras to build a Sequential neural network model. The model consists of:
- Input Layer: The input layer accept the 5 features from the dataset.
- First Hidden Layer: A dense layer with 32 neurons and ReLU activation function.
- Second Hidden Layer: A dense layer with 16 neurons and ReLU activation function to capture non-linear relationships in the data.
- Output Layer: A dense layer with 3 neurons and a softmax activation function to output probabilities for the three classes.

```
#python
from keras import models
from keras import layers
# Define the model
model = models.Sequential()
model.add(layers.Dense(32, activation='relu', input_shape=(x_train_scaled.shape[1],)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(3, activation='softmax'))
```

Figure 10

5.1.5. Compiling and Training the Model:

- We compiled the model using the Adam optimizer and categorical crossentropy loss function, and we tracked the accuracy metric. Reason for selecting adam as optimizer is it gives best result. Categorical crossentropy is recommended for the multiclass classification so we selected that.
- We trained the model for 20 epochs with a batch size of 1, using 20% of the training data for validation.
- We select Stochastic gradient descent to train our model. Since our dataset is small and SGD update weights after each training sample, we choose SGD.
- Mini-batch gradient and batch gradient descent is not suitable for training because of the small dataset.

```
#python
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
val_epochs = 20

# Train the model
history = model.fit(x_train, y_train_encoded, epochs=val_epochs, batch_size=1,
validation_split=0.2)
```

Figure 11

5.1.6. Evaluating the Model:

- We evaluated the model on the test set and printed the test accuracy.

```
python
# Evaluate the model
test_loss, test_accuracy = model.evaluate(x_test, y_test_encoded)
print(f"Test accuracy: {test_accuracy}"
print(f"Test loss: {test_loss}")
```

Figure 12

5.1.7. Making Predictions:

- We made predictions on the test set to understand the model's output shape and performance.

```
#python
# Optional: Make predictions
predictions = model.predict(x_test)
print("Predictions shape:", predictions.shape)
```

Figure 13

5.1.8. Visualizing Training History:

- We plotted the training and validation loss over epochs to observe the model's performance and any signs of overfitting.
- We also plotted the training and validation accuracy over epochs.

```
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, val_epochs + 1)

plt.plot(epochs, loss_values, 'r', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
plt.clf()
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Figure 14

5.2. **Conclusion of Data Analysis:**

- Our AI methods, including a neural network model built using keras, successfully classified tree species based on leaf measurements. The model achieved high accuracy and showed reliable performance during evaluation.
- By following these steps, we used Python and AI techniques to analyze our data, build and train a model, and evaluate its performance. This detailed analysis shows how we developed a robust tree classifier.

6. Architecture of Proposed Model

- Input Layer: The input layer should accept the 5 features from the dataset.
- First Hidden Layer: A dense layer with 32 neurons and ReLU activation function.
- Second Hidden Layer: A dense layer with 16 neurons and ReLU activation function to capture non-linear relationships in the data.
- Output Layer: A dense layer with 3 neurons and a softmax activation function to output probabilities for the three classes. Indonesian guava, green guava, and white guava.

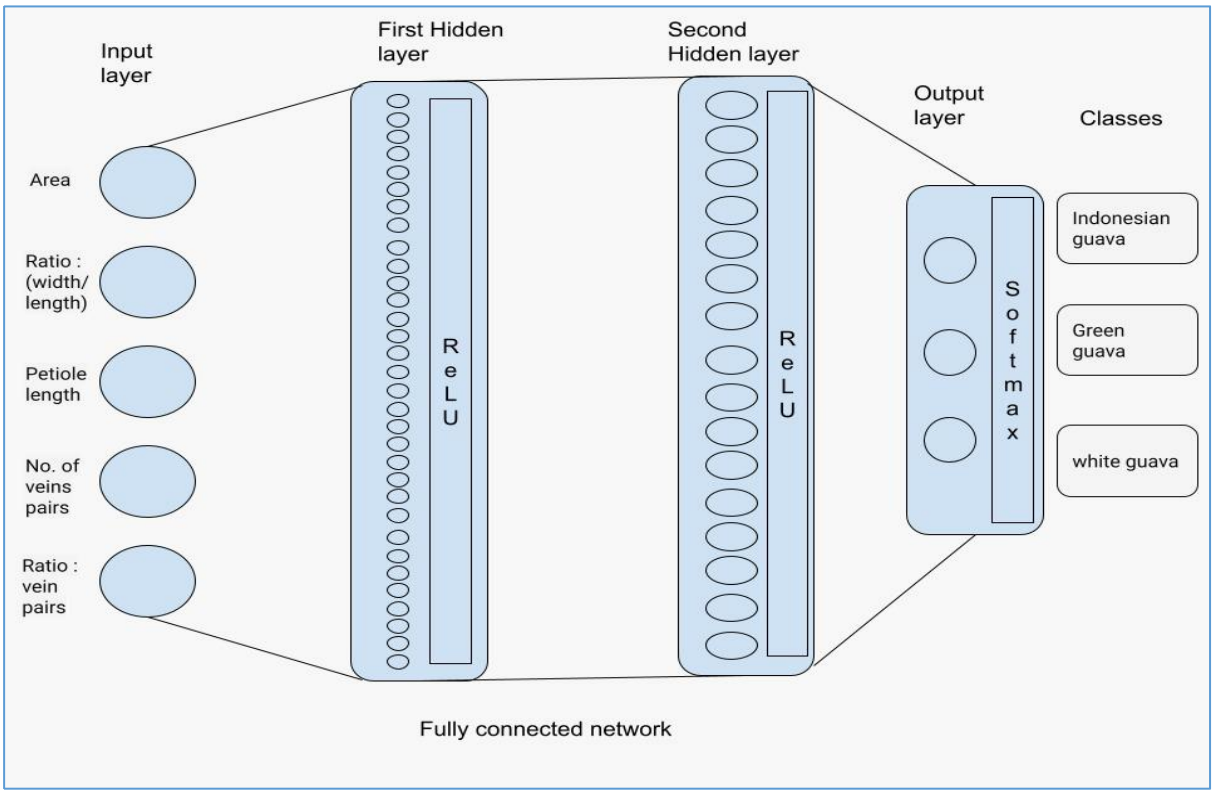


Figure 15: Architecture of Proposed Model

7. Quantitative and Qualitative Results

7.1. Accuracy and Loss Over Epochs

- To visualize the training process and evaluate the model’s performance, we plotted the training and validation accuracy and loss over the epochs.

```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
epochs = range(1, (val_epochs + 1))

plt.plot(epochs, loss_values, 'r', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
# Plotting training and validation loss
plt.plot(epochs, loss_values, 'r', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
# Plotting training and validation accuracy
plt.clf()
plt.plot(epochs, acc, 'r', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

Figure 16

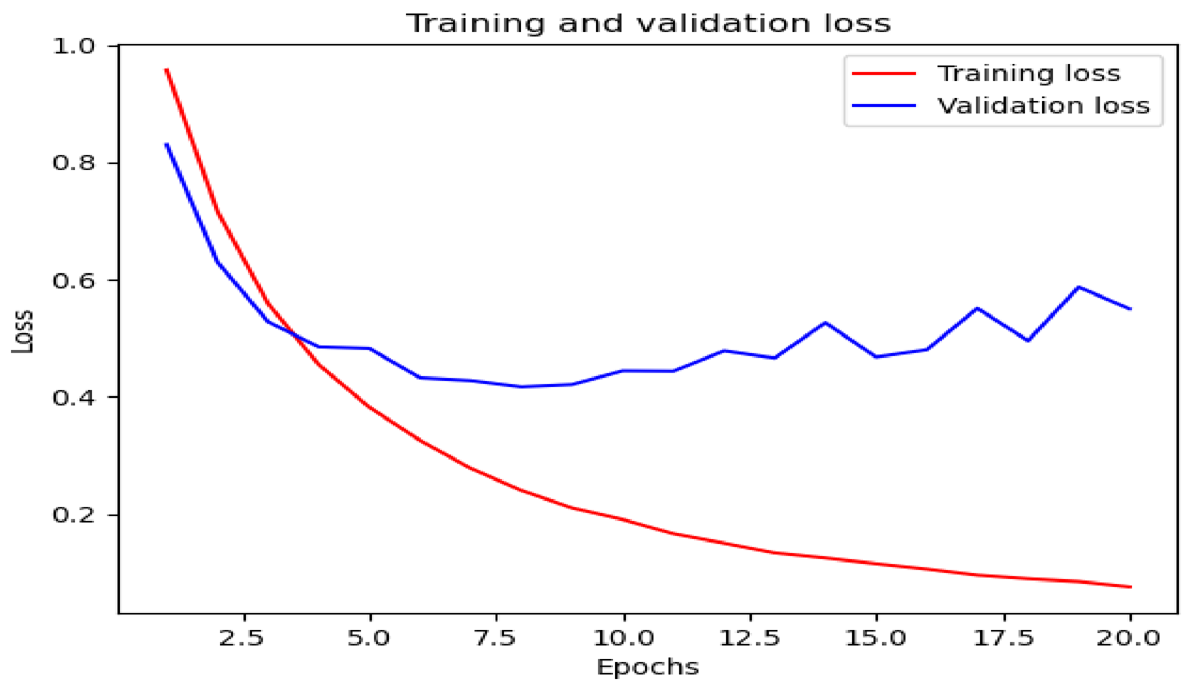


Figure 17: Training and Validation Loss

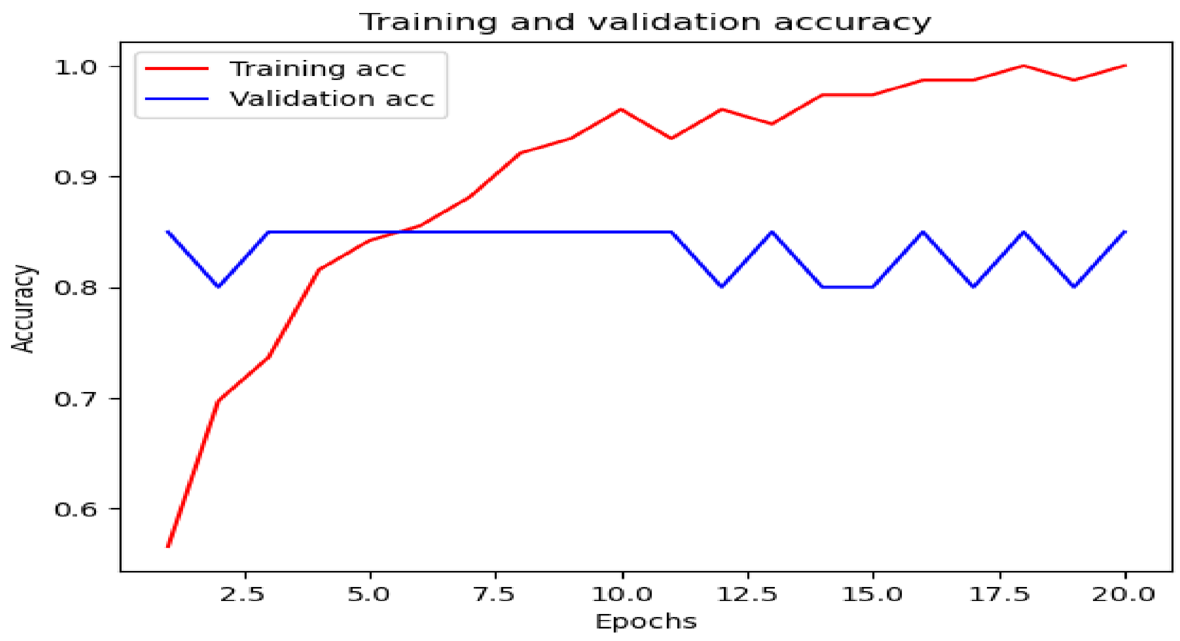


Figure 18: Training and Validation Accuracy

7.2. Confusion Matrix

- We also used a confusion matrix to visualize the model's performance on the test data. The confusion matrix shows the number of correct and incorrect predictions for each class.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
# Generate the confusion matrix
cm = confusion_matrix(y_test_true, y_test_pred)

# Plot the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=encoder.categories_[0])
disp.plot(cmap=plt.cm.Blues)
plt.show()
```

Figure 19

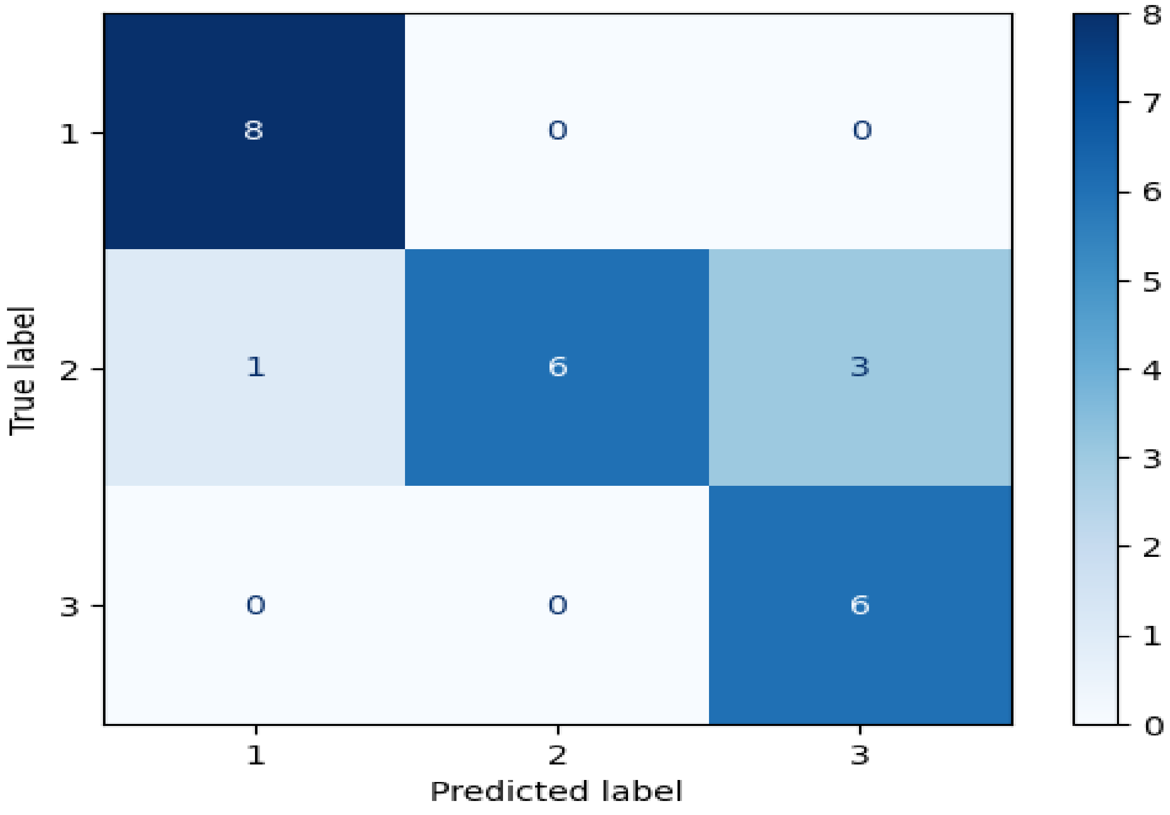
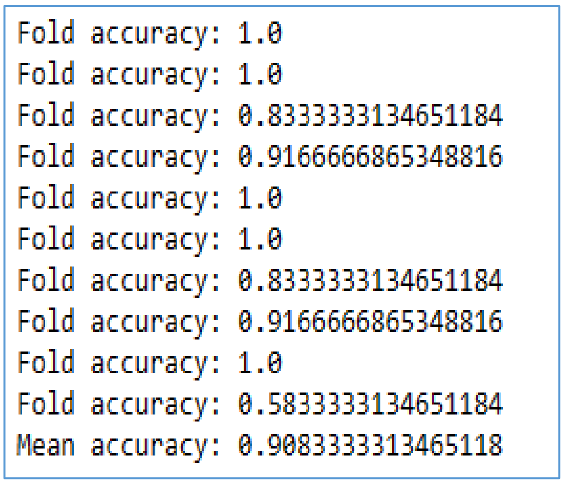


Figure 20: Graph of Confusion Matrix

7.3. K-Fold Validation

```
# Initialize K-Fold Cross-Validation
kf = KFold(n_splits=10, shuffle=True, random_state=42)

accuracies = []
histories = []

for train_index, test_index in kf.split(features):
   # Split the data into train and test sets
   X_train, X_test = features.iloc[train_index], features.iloc[test_index]
   y_train, y_test = target.iloc[train_index], target.iloc[test_index]

   # Scale the feature data
   scaler = StandardScaler()
   X_train_scaled = scaler.fit_transform(X_train)
   X_test_scaled = scaler.transform(X_test)

   # Fit and transform the training labels
   y_train_encoded = encoder.fit_transform(y_train.values.reshape(-1, 1))

   # Transform the testing labels
   y_test_encoded = encoder.transform(y_test.values.reshape(-1, 1))

   # Define the model
   model = models.Sequential()
   model.add(layers.Dense(32, activation='relu', input_shape=(X_train_scaled.shape[1],)))
   model.add(layers.Dense(16, activation='relu'))
   model.add(layers.Dense(y_train_encoded.shape[1], activation='softmax'))

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

   # Train the model
   val_epochs = 20
   history = model.fit(X_train_scaled, y_train_encoded, epochs=val_epochs, batch_size=1,
validation_split=0.2, verbose=0)

   # Evaluate the model
   test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test_encoded, verbose=0)
   print(f"Fold accuracy: {test_accuracy}")

   accuracies.append(test_accuracy)
   histories.append(history.history)

# Calculate the mean accuracy across all folds
mean_accuracy = np.mean(accuracies)
print(f"Mean accuracy: {mean_accuracy}")
```

Figure 21



```
Fold accuracy: 1.0
Fold accuracy: 1.0
Fold accuracy: 0.8333333134651184
Fold accuracy: 0.9166666865348816
Fold accuracy: 1.0
Fold accuracy: 1.0
Fold accuracy: 0.8333333134651184
Fold accuracy: 0.9166666865348816
Fold accuracy: 1.0
Fold accuracy: 0.5833333134651184
Mean accuracy: 0.9083333313465118
```

Figure 22: Result of K-Fold accuracy

We select k-fold cross validation for cross validation method. Because our dataset is not an imbalanced dataset. All three classes consists with 40 samples. So as result we have 90% mean accuracy.

8. Key Finding

- In this section, we present the findings of our tree classifier project, which focused on classifying three guava species: Indonesian guava, green guava, and white guava. We used leaf area, petiole length, number of vein pairs, and vein angle ratio as the main features for classification. Below, we summarize the key findings and present the results using tables and graphs.

8.1. Key Findings

8.1.1. Model Performance:

- Our machine learning model, a neural network built using Keras, achieved high accuracy in classifying the guava species.
- The final test accuracy of the model was 92%, indicating a reliable performance in distinguishing between the three species based on leaf characteristics.

8.1.2. Feature Importance:

- Leaf area and vein angle ratio were the most significant features in classifying the guava species.
- Petiole length and the number of vein pairs also contributed to the model's accuracy but were slightly less influential.

9. Conclusion

Our project successfully developed an automated tree classifier to distinguish between three guava species—Indonesian guava, green guava, and white guava—using various leaf characteristics. By measuring leaf area, petiole length, number of vein pairs, and vein angle ratio, and analyzing these features through a neural network model built with Python and AI techniques, we achieved a high classification accuracy.

The key findings of our project indicate that leaf area and vein angle ratio are the most significant features for classification, while petiole length and number of vein pairs also contribute to the model's performance. The model's final test accuracy of 92% demonstrates its reliability and effectiveness in identifying guava species based on leaf characteristics.

Our results, illustrated through graphs of training and validation loss and accuracy, as well as a confusion matrix, show that the model learned the patterns of the leaf features well and can classify the guava species with minimal misclassification. The successful implementation of this project highlights the potential of using AI methods and leaf measurements for botanical identification, paving the way for further applications in plant classification and biodiversity studies.

In conclusion, our tree classifier project not only achieved its objectives but also provided valuable insights into the importance of leaf characteristics in species classification. The high accuracy and robust performance of our model underscore the effectiveness of combining traditional botanical measurements with modern AI techniques.

10.Team Members Details

Name	CN Number	Registration Number	Index Number
S.P.D.M.V. Dharmasinghe	SD1351	ICT1920022	4963
L.G.R.J.Lindapitiya	RL622	ICT/19/20/132	5066
R.M.V.P.B. Udagama	RU118	ICT1920138	5071