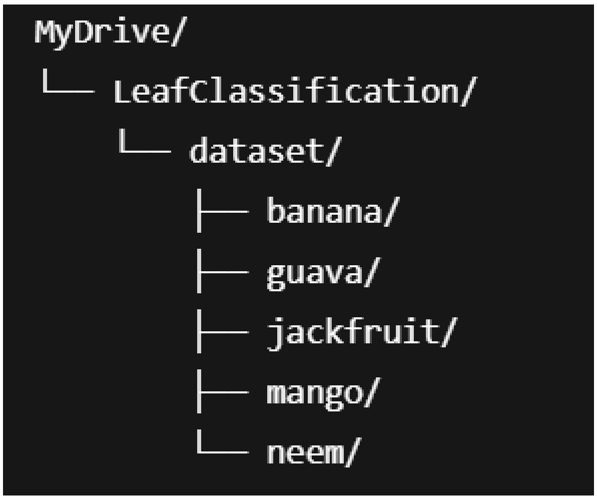
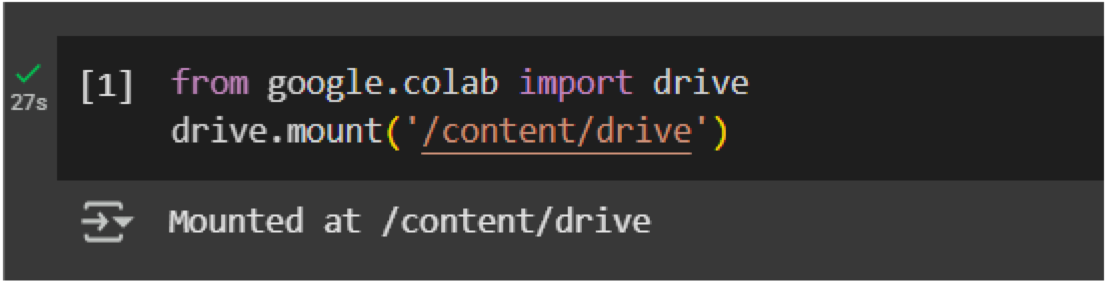
1. Collect Dataset of 5 leaves (Mango, Guava, Jackfruit, Neem, Tulsi) 500 each.
2. Upload in drive



```
MyDrive/
└── LeafClassification/
    └── dataset/
        ├── banana/
        ├── guava/
        ├── jackfruit/
        ├── mango/
        └── neem/
```

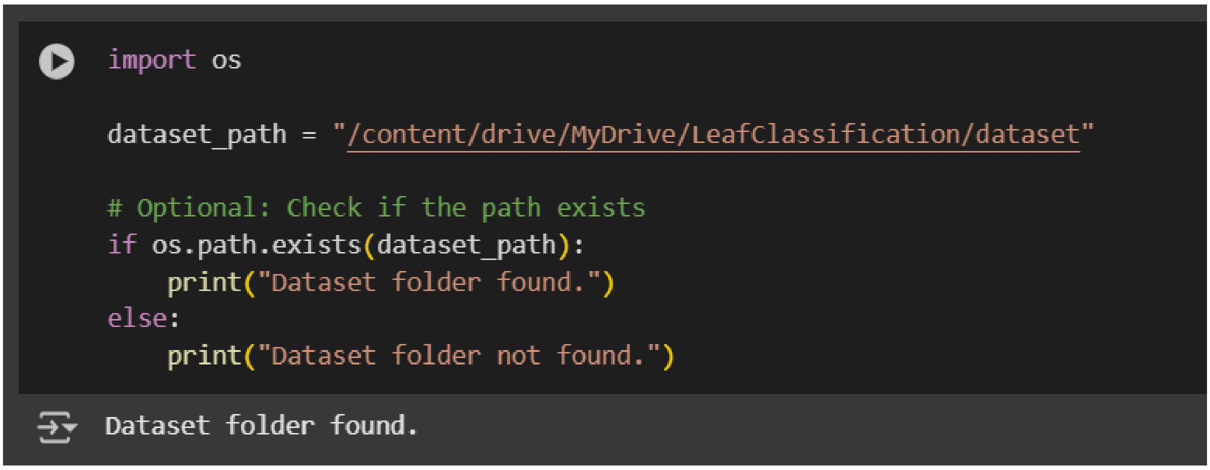
3. Open co-lab and Execute step by step.



```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

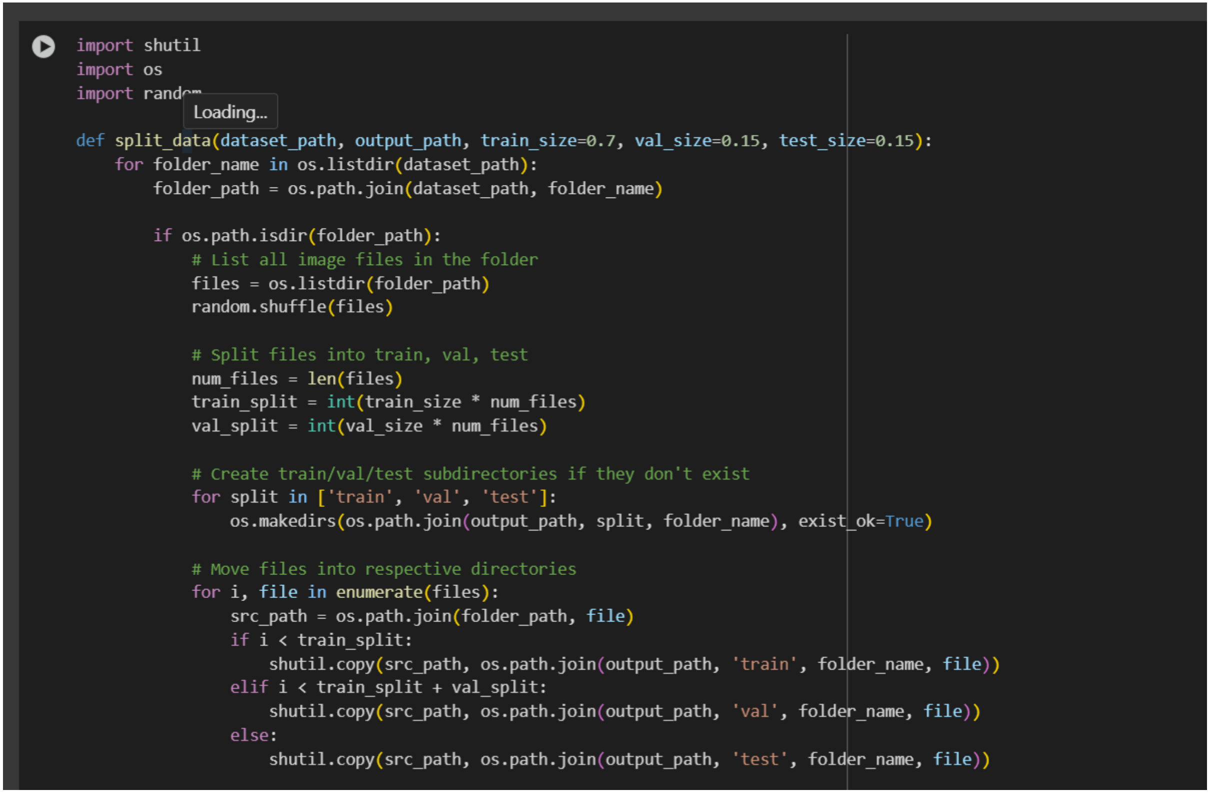
4. Find dataset.



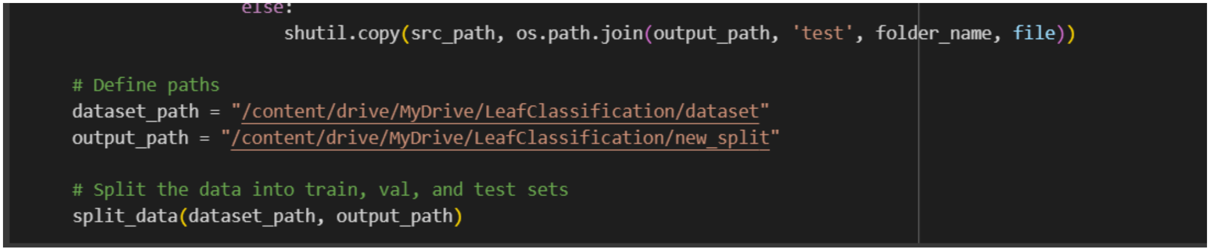
```
import os

dataset_path = "/content/drive/MyDrive/LeafClassification/dataset"

# Optional: Check if the path exists
if os.path.exists(dataset_path):
    print("Dataset folder found.")
else:
    print("Dataset folder not found.")
```

```
Dataset folder found.
```

5. Split data in new folder.



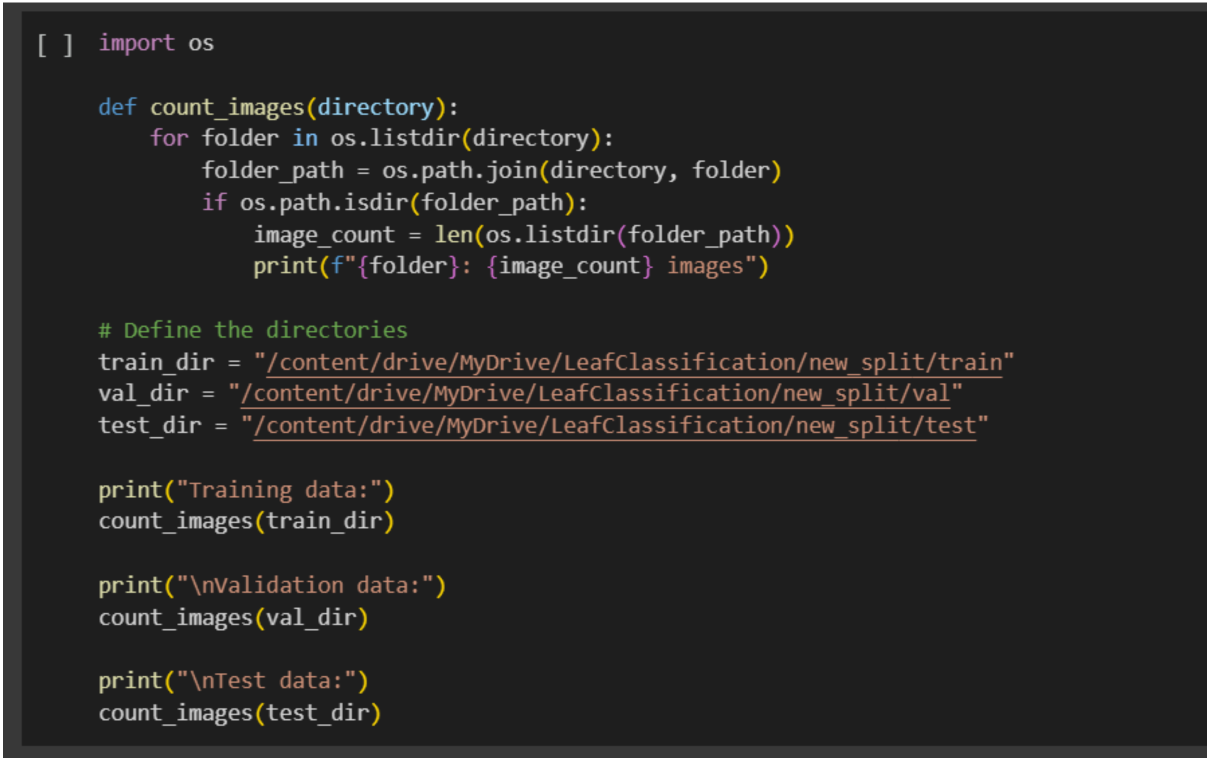
```python
import shutil
import os
import random

def split_data(dataset_path, output_path, train_size=0.7, val_size=0.15, test_size=0.15):
    for folder_name in os.listdir(dataset_path):
        folder_path = os.path.join(dataset_path, folder_name)

        if os.path.isdir(folder_path):
            # List all image files in the folder
            files = os.listdir(folder_path)
            random.shuffle(files)

            # Split files into train, val, test
            num_files = len(files)
            train_split = int(train_size * num_files)
            val_split = int(val_size * num_files)

            # Create train/val/test subdirectories if they don't exist
            for split in ['train', 'val', 'test']:
                os.makedirs(os.path.join(output_path, split, folder_name), exist_ok=True)

            # Move files into respective directories
            for i, file in enumerate(files):
                src_path = os.path.join(folder_path, file)
                if i < train_split:
                    shutil.copy(src_path, os.path.join(output_path, 'train', folder_name, file))
                elif i < train_split + val_split:
                    shutil.copy(src_path, os.path.join(output_path, 'val', folder_name, file))
                else:
                    shutil.copy(src_path, os.path.join(output_path, 'test', folder_name, file))
```



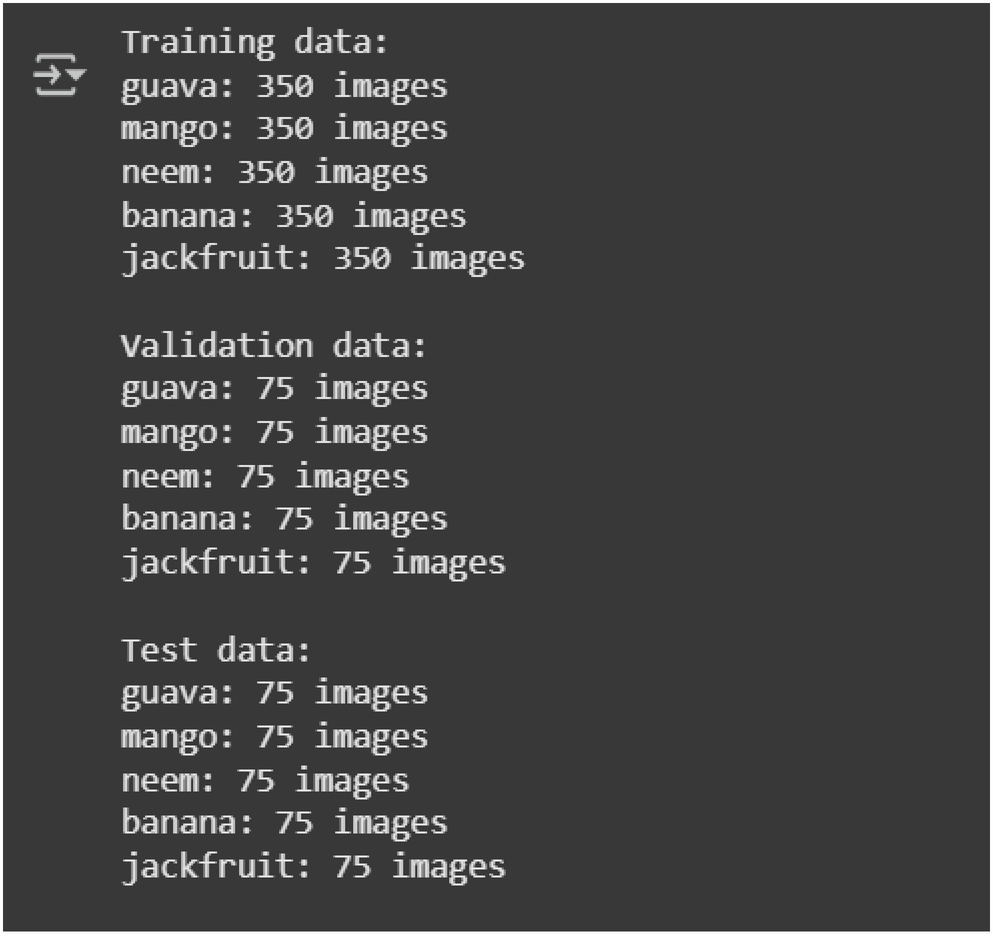
```python
                else:
                    shutil.copy(src_path, os.path.join(output_path, 'test', folder_name, file))
```

```python
# Define paths
dataset_path = "/content/drive/MyDrive/LeafClassification/dataset"
output_path = "/content/drive/MyDrive/LeafClassification/new_split"

# Split the data into train, val, and test sets
split_data(dataset_path, output_path)
```

6. View split data.

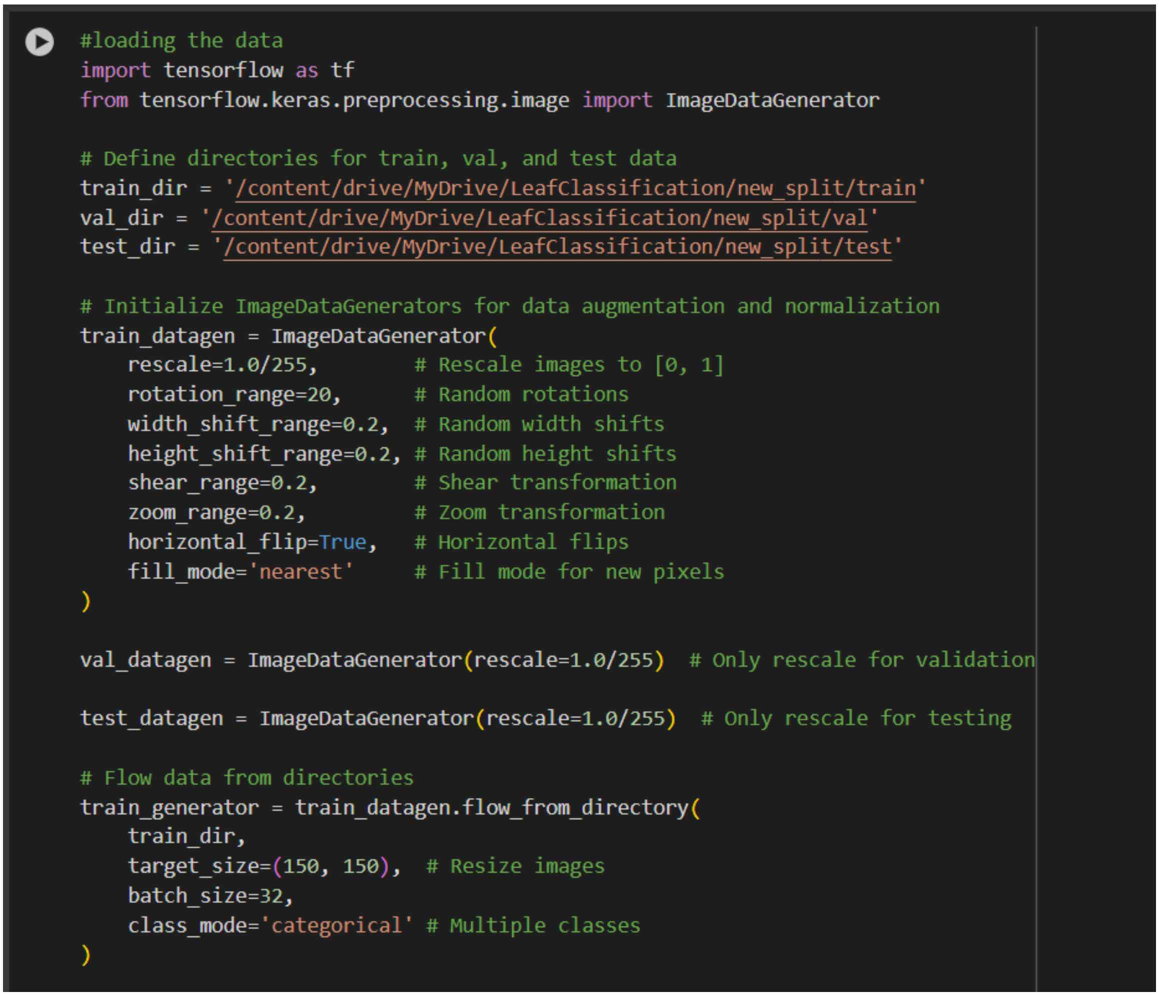


```
import os

def count_images(directory):
    for folder in os.listdir(directory):
        folder_path = os.path.join(directory, folder)
        if os.path.isdir(folder_path):
            image_count = len(os.listdir(folder_path))
            print(f"{folder}: {image_count} images")

# Define the directories
train_dir = "/content/drive/MyDrive/LeafClassification/new_split/train"
val_dir = "/content/drive/MyDrive/LeafClassification/new_split/val"
test_dir = "/content/drive/MyDrive/LeafClassification/new_split/test"

print("Training data:")
count_images(train_dir)

print("\nValidation data:")
count_images(val_dir)

print("\nTest data:")
count_images(test_dir)
```

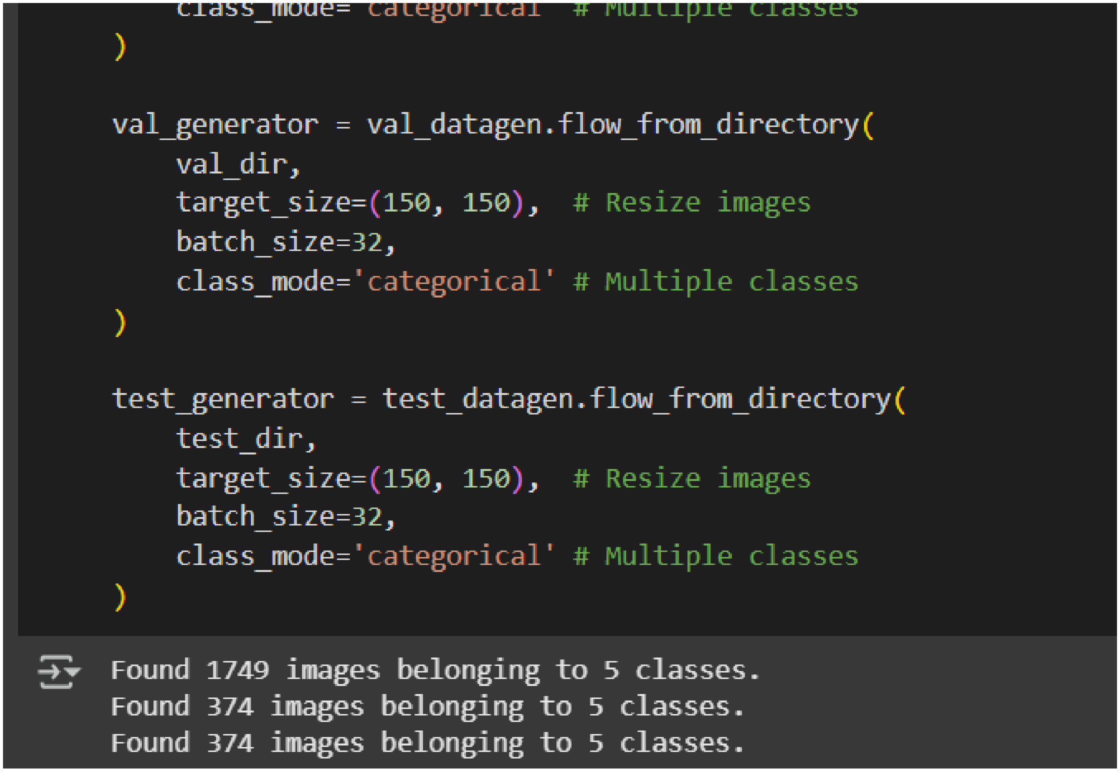


```
Training data:
guava: 350 images
mango: 350 images
neem: 350 images
banana: 350 images
jackfruit: 350 images

Validation data:
guava: 75 images
mango: 75 images
neem: 75 images
banana: 75 images
jackfruit: 75 images

Test data:
guava: 75 images
mango: 75 images
neem: 75 images
banana: 75 images
jackfruit: 75 images
```

7. Load Data.



```python
#loading the data
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define directories for train, val, and test data
train_dir = '/content/drive/MyDrive/LeafClassification/new_split/train'
val_dir = '/content/drive/MyDrive/LeafClassification/new_split/val'
test_dir = '/content/drive/MyDrive/LeafClassification/new_split/test'

# Initialize ImageDataGenerators for data augmentation and normalization
train_datagen = ImageDataGenerator(
    rescale=1.0/255,         # Rescale images to [0, 1]
    rotation_range=20,       # Random rotations
    width_shift_range=0.2,   # Random width shifts
    height_shift_range=0.2,  # Random height shifts
    shear_range=0.2,         # Shear transformation
    zoom_range=0.2,          # Zoom transformation
    horizontal_flip=True,    # Horizontal flips
    fill_mode='nearest'      # Fill mode for new pixels
)

val_datagen = ImageDataGenerator(rescale=1.0/255)  # Only rescale for validation

test_datagen = ImageDataGenerator(rescale=1.0/255)  # Only rescale for testing

# Flow data from directories
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),  # Resize images
    batch_size=32,
    class_mode='categorical' # Multiple classes
)
```

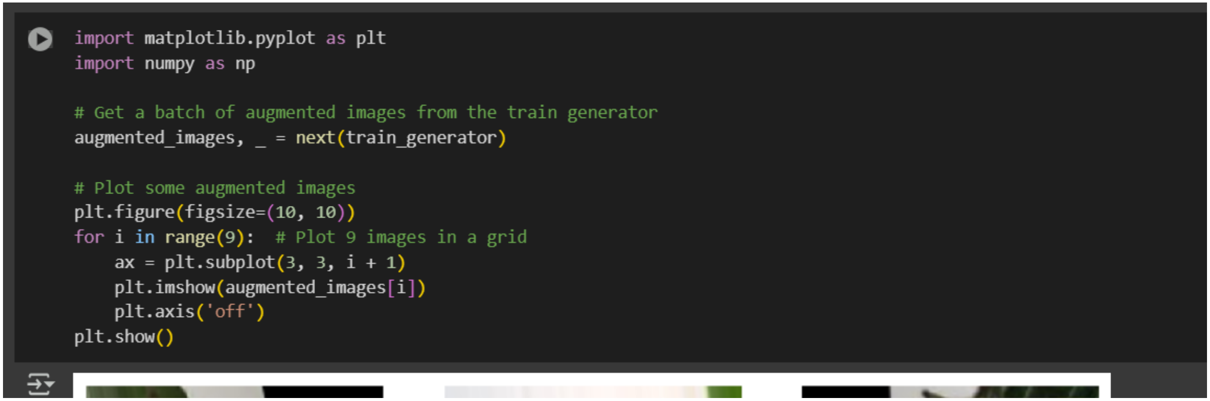


```python
    class_mode='categorical' # Multiple classes
)
```

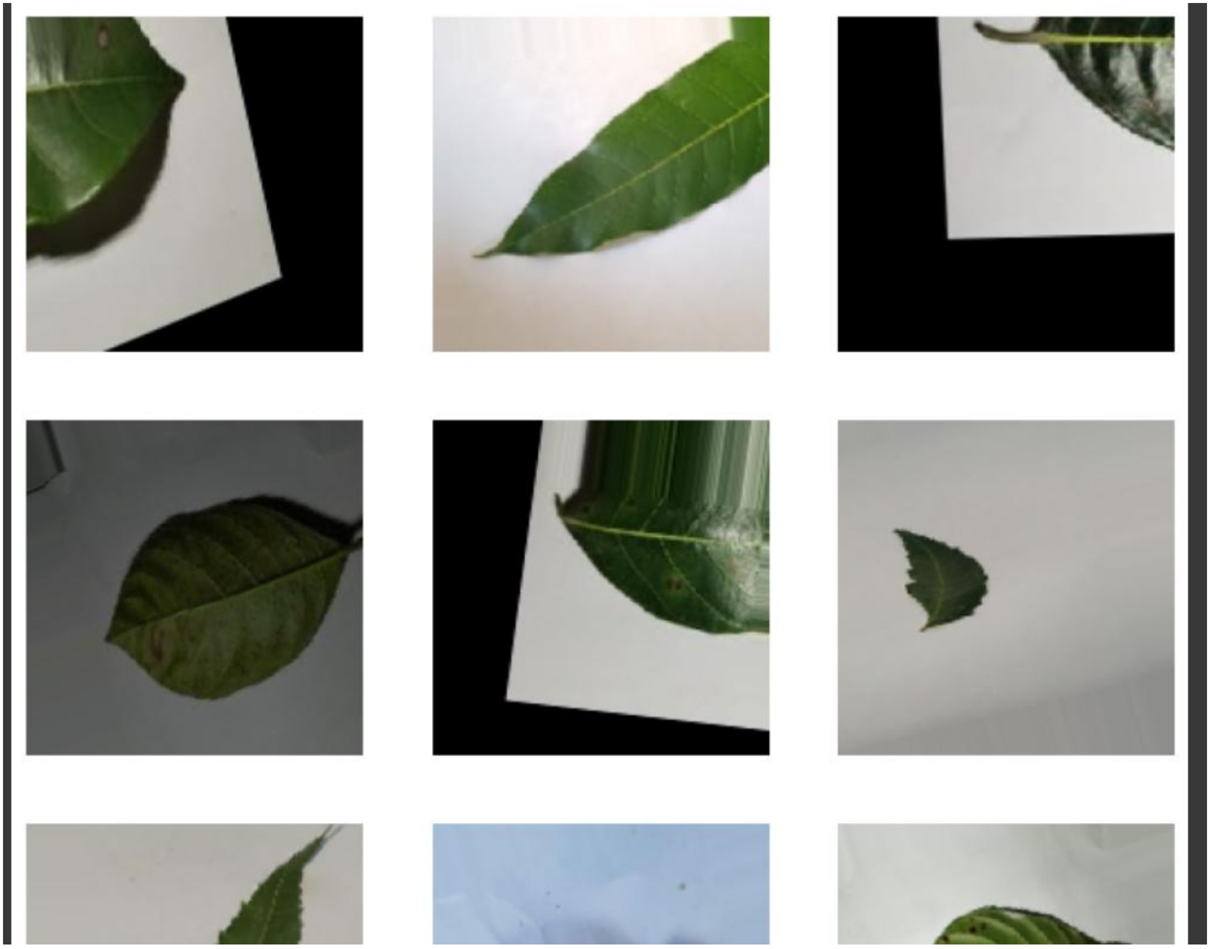
```python
val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(150, 150),  # Resize images
    batch_size=32,
    class_mode='categorical' # Multiple classes
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),  # Resize images
    batch_size=32,
    class_mode='categorical' # Multiple classes
)
```

```
Found 1749 images belonging to 5 classes.
Found 374 images belonging to 5 classes.
Found 374 images belonging to 5 classes.
```

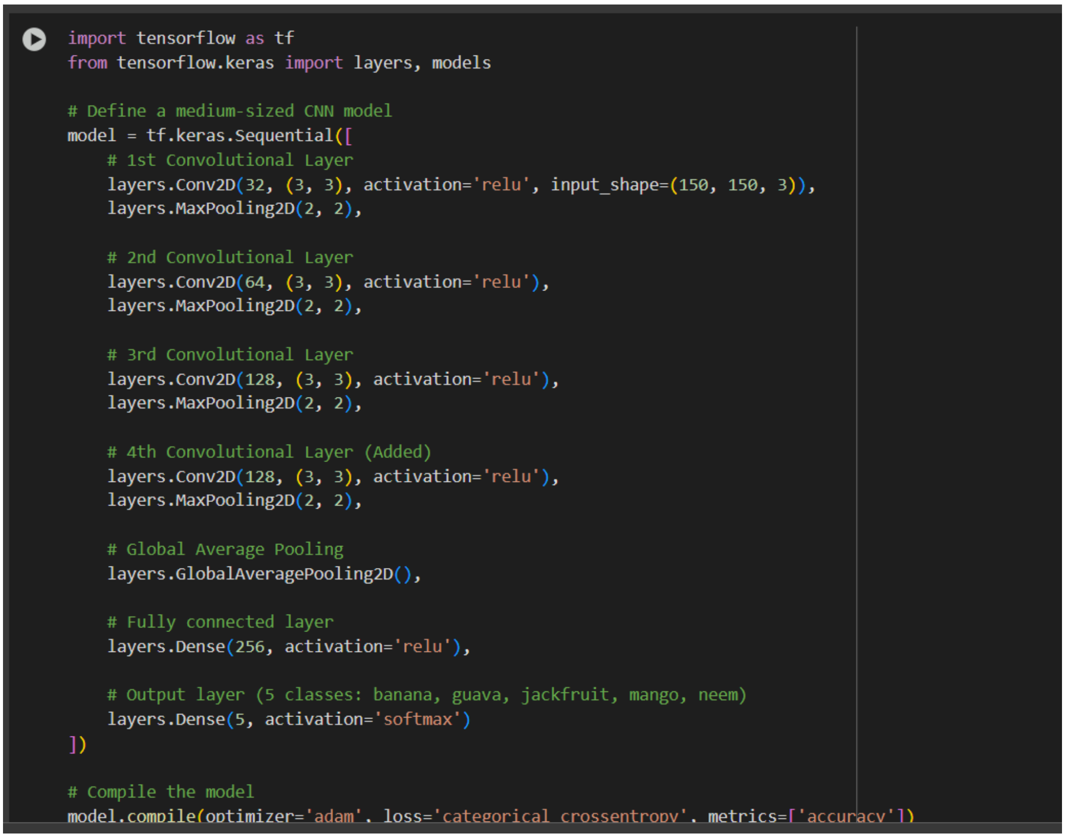
8. Verify augmentation.



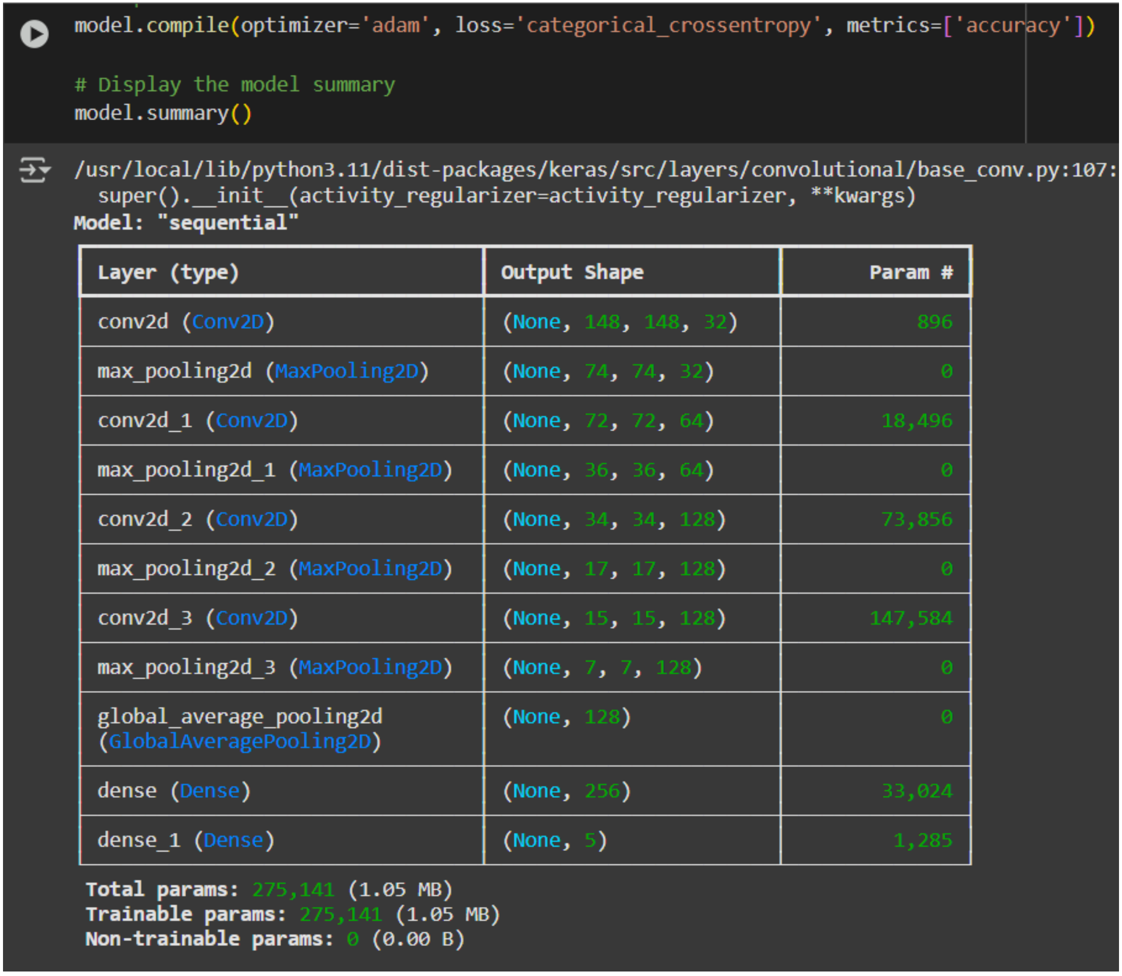
```
import matplotlib.pyplot as plt
import numpy as np

# Get a batch of augmented images from the train generator
augmented_images, _ = next(train_generator)

# Plot some augmented images
plt.figure(figsize=(10, 10))
for i in range(9):  # Plot 9 images in a grid
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[i])
    plt.axis('off')
plt.show()
```



9. Model



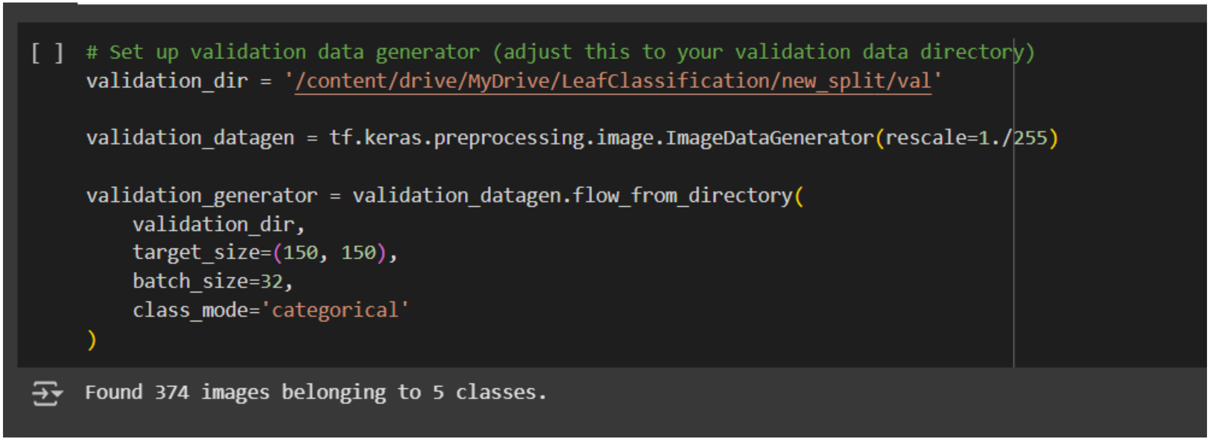
```
import tensorflow as tf
from tensorflow.keras import layers, models

# Define a medium-sized CNN model
model = tf.keras.Sequential([
    # 1st Convolutional Layer
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    layers.MaxPooling2D(2, 2),

    # 2nd Convolutional Layer
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    # 3rd Convolutional Layer
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    # 4th Convolutional Layer (Added)
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D(2, 2),

    # Global Average Pooling
    layers.GlobalAveragePooling2D(),

    # Fully connected layer
    layers.Dense(256, activation='relu'),

    # Output layer (5 classes: banana, guava, jackfruit, mango, neem)
    layers.Dense(5, activation='softmax')
])

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



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

# Display the model summary
model.summary()
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107:
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                     ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                  │ (None, 148, 148, 32)   │           896 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)     │ (None, 74, 74, 32)     │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)                │ (None, 72, 72, 64)     │        18,496 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)   │ (None, 36, 36, 64)     │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)                │ (None, 34, 34, 128)    │        73,856 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_2 (MaxPooling2D)   │ (None, 17, 17, 128)    │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_3 (Conv2D)                │ (None, 15, 15, 128)    │       147,584 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_3 (MaxPooling2D)   │ (None, 7, 7, 128)      │             0 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ global_average_pooling2d         │ (None, 128)            │             0 │
│ (GlobalAveragePooling2D)         │                        │               │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                    │ (None, 256)            │        33,024 │
├──────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                  │ (None, 5)              │         1,285 │
└──────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 275,141 (1.05 MB)
 Trainable params: 275,141 (1.05 MB)
 Non-trainable params: 0 (0.00 B)
```

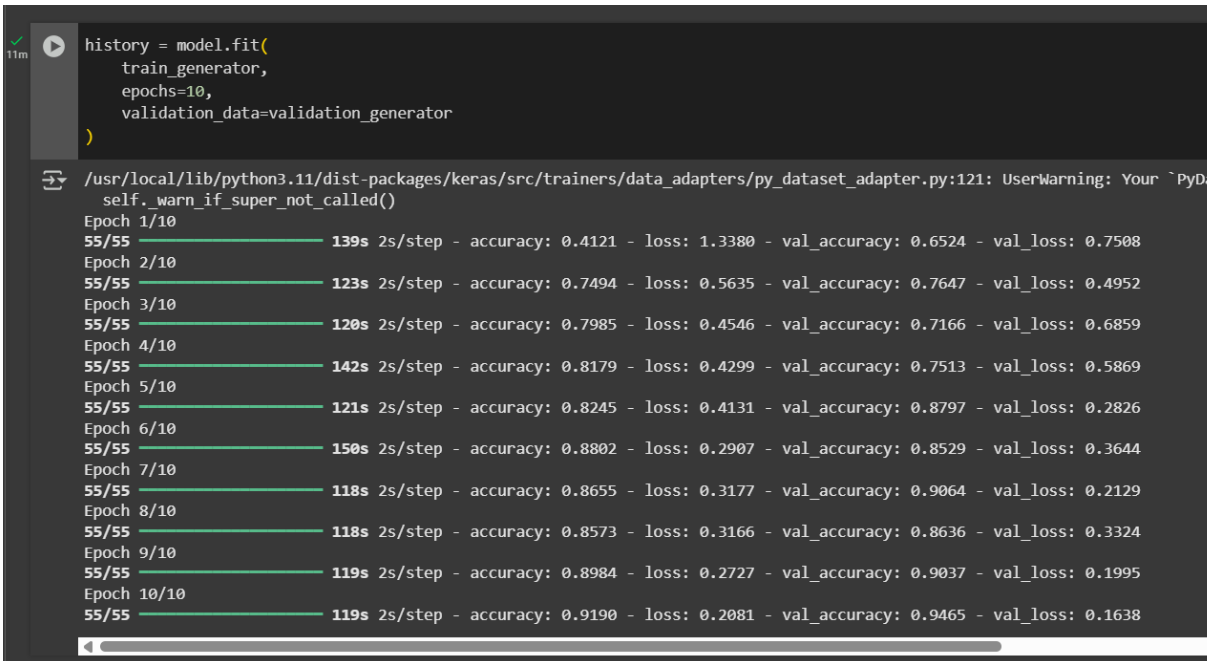
10. Validation Data generator.



```
# Set up validation data generator (adjust this to your validation data directory)
validation_dir = '/content/drive/MyDrive/LeafClassification/new_split/val'

validation_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)

validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)
```

```
Found 374 images belonging to 5 classes.
```

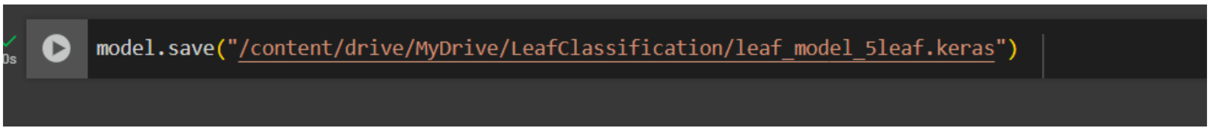
11. Model fit.



```
history = model.fit(
    train_generator,
    epochs=10,
    validation_data=validation_generator
)
```

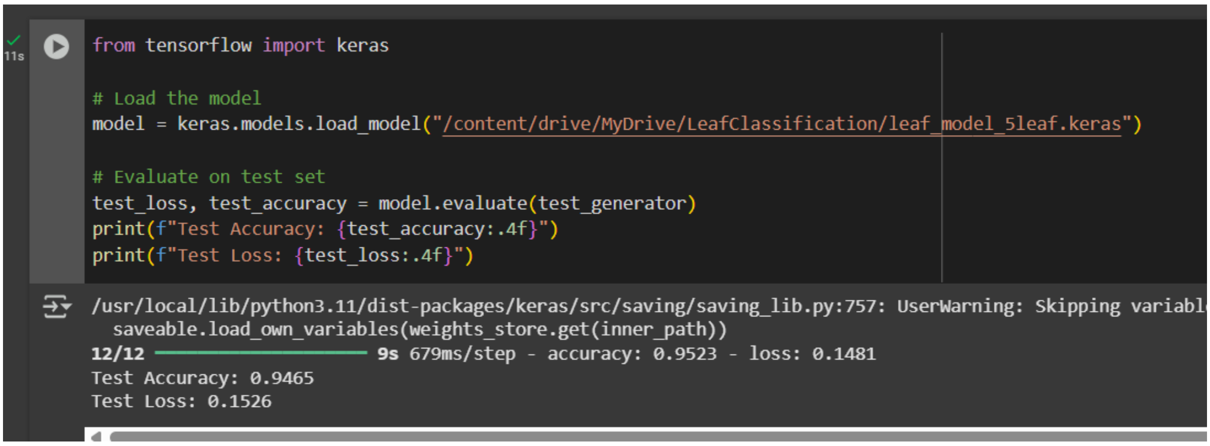
```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyD
  self._warn_if_super_not_called()
Epoch 1/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 139s 2s/step - accuracy: 0.4121 - loss: 1.3380 - val_accuracy: 0.6524 - val_loss: 0.7508
Epoch 2/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 123s 2s/step - accuracy: 0.7494 - loss: 0.5635 - val_accuracy: 0.7647 - val_loss: 0.4952
Epoch 3/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 120s 2s/step - accuracy: 0.7985 - loss: 0.4546 - val_accuracy: 0.7166 - val_loss: 0.6859
Epoch 4/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.8179 - loss: 0.4299 - val_accuracy: 0.7513 - val_loss: 0.5869
Epoch 5/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 121s 2s/step - accuracy: 0.8245 - loss: 0.4131 - val_accuracy: 0.8797 - val_loss: 0.2826
Epoch 6/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 150s 2s/step - accuracy: 0.8802 - loss: 0.2907 - val_accuracy: 0.8529 - val_loss: 0.3644
Epoch 7/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 118s 2s/step - accuracy: 0.8655 - loss: 0.3177 - val_accuracy: 0.9064 - val_loss: 0.2129
Epoch 8/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 118s 2s/step - accuracy: 0.8573 - loss: 0.3166 - val_accuracy: 0.8636 - val_loss: 0.3324
Epoch 9/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 119s 2s/step - accuracy: 0.8984 - loss: 0.2727 - val_accuracy: 0.9037 - val_loss: 0.1995
Epoch 10/10
55/55 ━━━━━━━━━━━━━━━━━━━━ 119s 2s/step - accuracy: 0.9190 - loss: 0.2081 - val_accuracy: 0.9465 - val_loss: 0.1638
```

12. Save model.



```
model.save("/content/drive/MyDrive/LeafClassification/leaf_model_5leaf.keras")
```

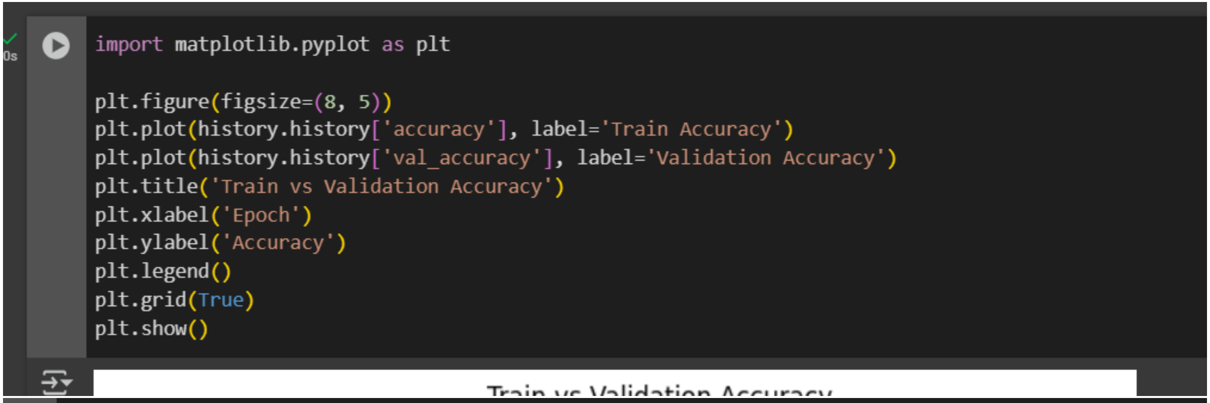
13. Test.



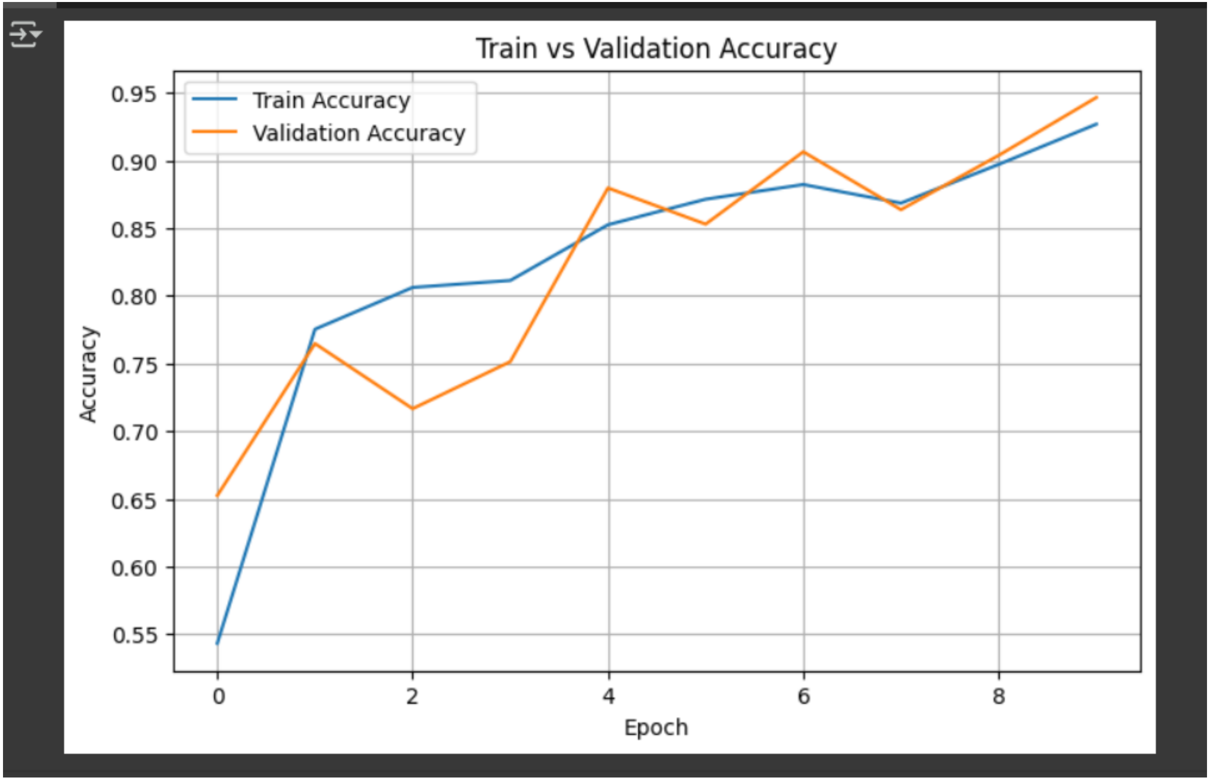
```
from tensorflow import keras

# Load the model
model = keras.models.load_model("/content/drive/MyDrive/LeafClassification/leaf_model_5leaf.keras")

# Evaluate on test set
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test Accuracy: {test_accuracy:.4f}")
print(f"Test Loss: {test_loss:.4f}")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/saving/saving_lib.py:757: UserWarning: Skipping variabl
  saveable.load_own_variables(weights_store.get(inner_path))
12/12 ━━━━━━━━━━━━━━━━━━━━ 9s 679ms/step - accuracy: 0.9523 - loss: 0.1481
Test Accuracy: 0.9465
Test Loss: 0.1526
```

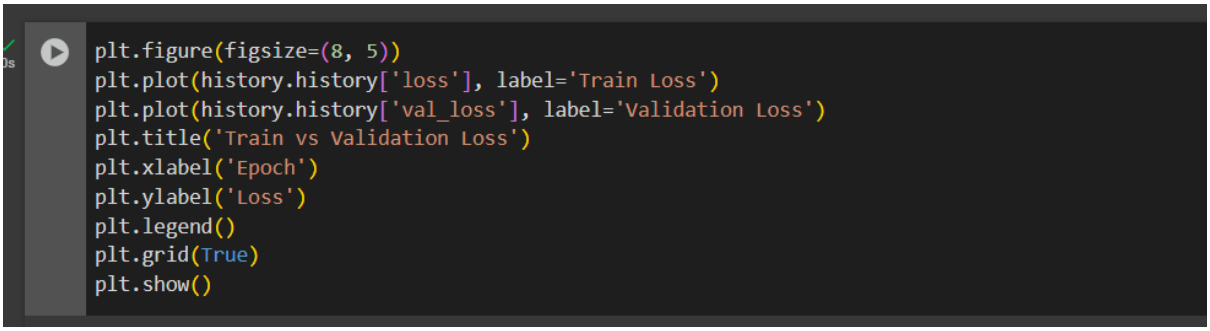
14. Train & Test accuracy curve.



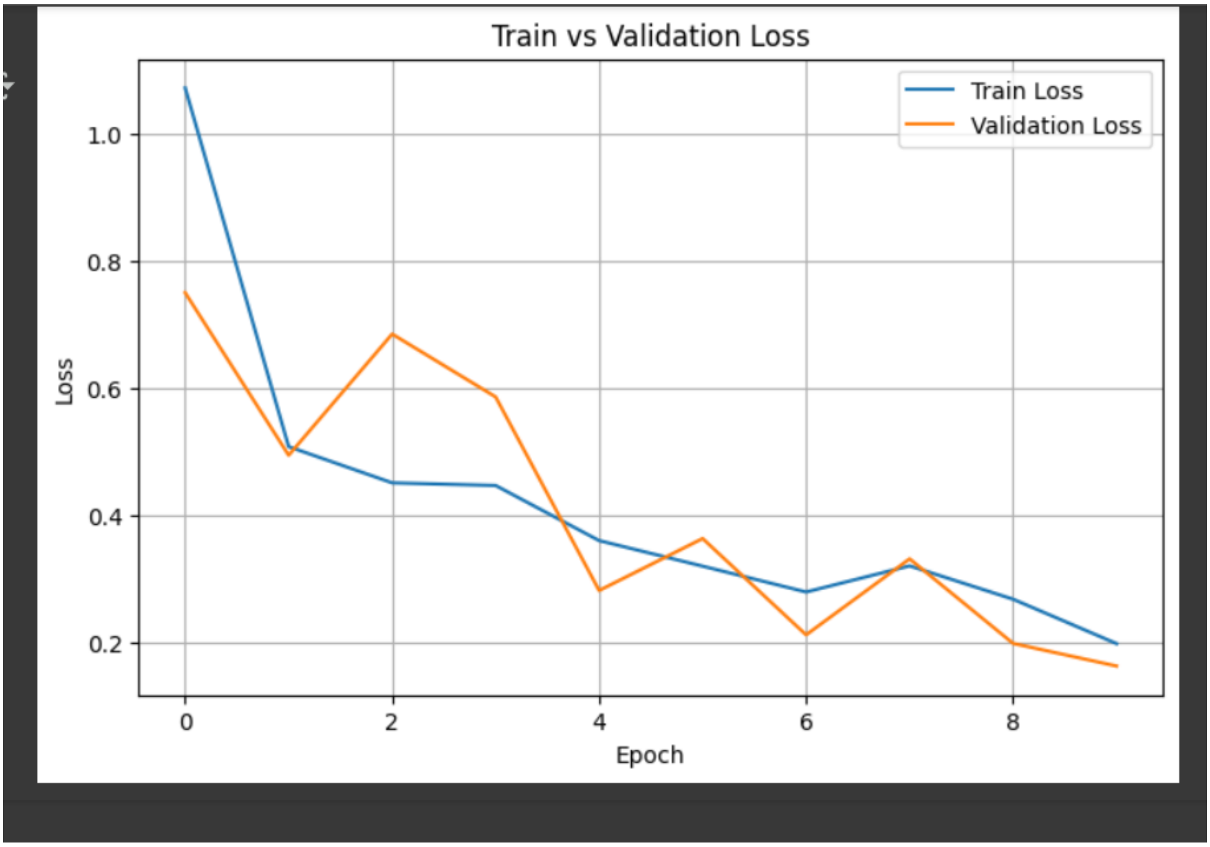
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Train vs Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.grid(True)
plt.show()
```



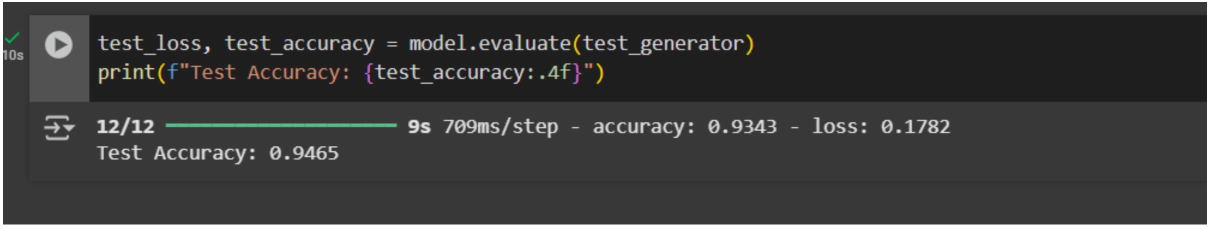
15. Train & Validation loss curve.



```
plt.figure(figsize=(8, 5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Train vs Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```



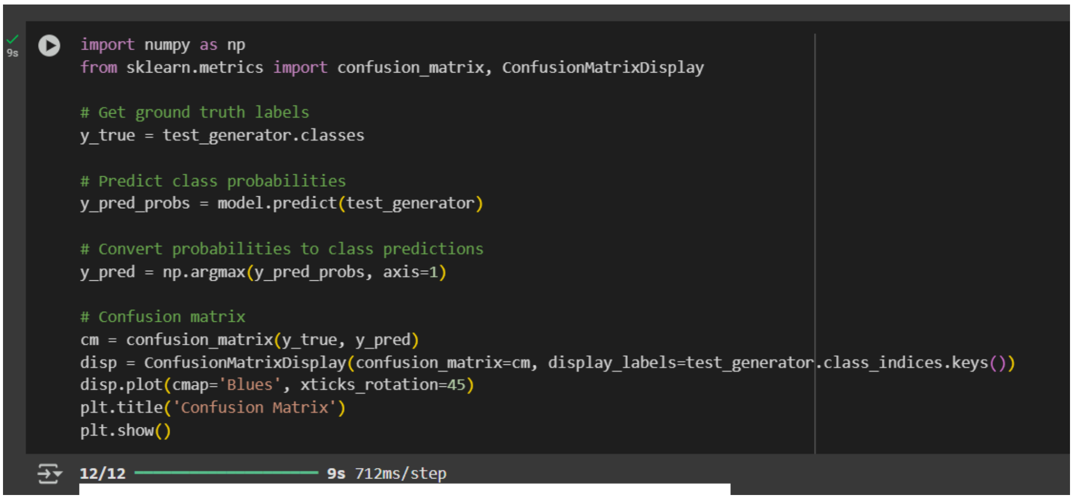
16. Test accuracy.



```
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

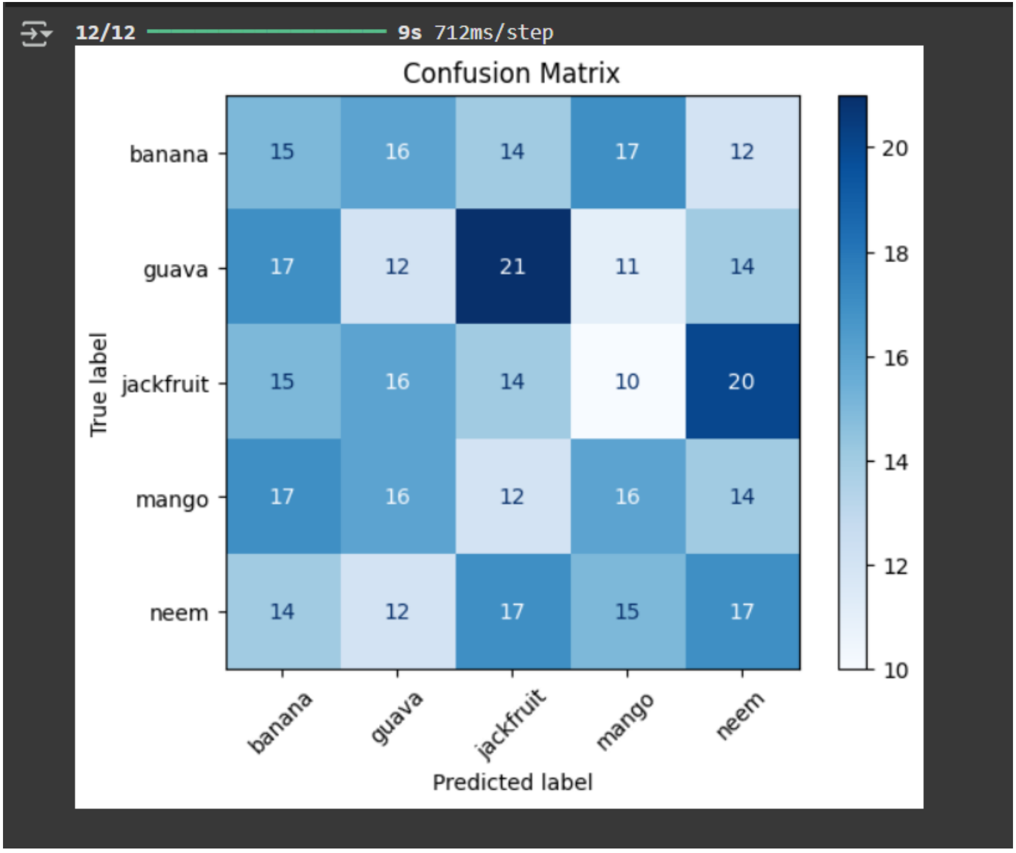
```
12/12 ━━━━━━━━━━━━━━━━━━━━ 9s 709ms/step - accuracy: 0.9343 - loss: 0.1782
Test Accuracy: 0.9465
```

17. Confusion Matrix.



```
import numpy as np
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Get ground truth labels
y_true = test_generator.classes

# Predict class probabilities
y_pred_probs = model.predict(test_generator)

# Convert probabilities to class predictions
y_pred = np.argmax(y_pred_probs, axis=1)

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=test_generator.class_indices.keys())
disp.plot(cmap='Blues', xticks_rotation=45)
plt.title('Confusion Matrix')
plt.show()
```

```
12/12 ━━━━━━━━━━━━━━━━━━━━ 9s 712ms/step
```

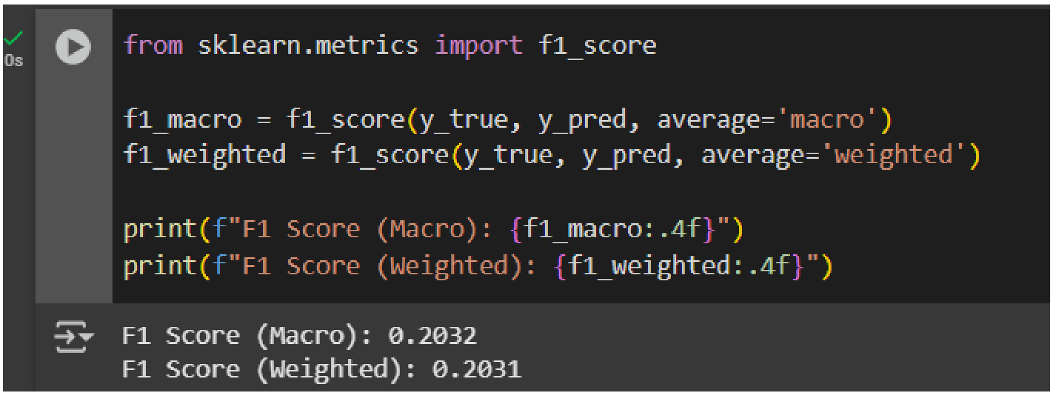


```
12/12 ━━━━━━━━━━━━━━━━━━━━ 9s 712ms/step
```

Confusion Matrix

	banana	guava	jackfruit	mango	neem
banana	15	16	14	17	12
guava	17	12	21	11	14
jackfruit	15	16	14	10	20
mango	17	16	12	16	14
neem	14	12	17	15	17

18. F1 Score.



```
from sklearn.metrics import f1_score

f1_macro = f1_score(y_true, y_pred, average='macro')
f1_weighted = f1_score(y_true, y_pred, average='weighted')

print(f"F1 Score (Macro): {f1_macro:.4f}")
print(f"F1 Score (Weighted): {f1_weighted:.4f}")
```

```
F1 Score (Macro): 0.2032
F1 Score (Weighted): 0.2031
```