```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping


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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Define paths to your datasets
train_dir = '/content/drive/MyDrive/Data/Train'
test_dir = '/content/drive/MyDrive/Data/Test'
val_dir = '/content/drive/MyDrive/Data/Validation'


# Image preprocessing
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=40,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
                                   fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)


# Loading and augmenting images
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical')

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical')

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical')
```

```
Found 1351 images belonging to 3 classes.
Found 150 images belonging to 3 classes.
Found 60 images belonging to 3 classes.
```

```
import os
import matplotlib.pyplot as plt
from PIL import Image

# Visualization function
def visualize_data(train_dir, categories, num_images=10, images_per_row=5):
    rows = len(categories) * (num_images // images_per_row)  # Calculate total rows
    plt.figure(figsize=(20, rows * 3))  # Adjust figure size based on rows

    img_index = 1  # For subplot index
    for category in categories:
        category_path = os.path.join(train_dir, category)  # Path to category folder
        image_files = os.listdir(category_path)[:num_images]  # Get first 'num_images' files

        for img_file in image_files:
            img_path = os.path.join(category_path, img_file)
            img = Image.open(img_path)

            # Subplot
            plt.subplot(rows, images_per_row, img_index)
            plt.imshow(img)
            plt.axis('off')
            if img_index % images_per_row == 1:  # Add category title only at the start of a row
                plt.title(category, fontsize=14, loc='left')
            img_index += 1

    plt.tight_layout()
```

```
    plt.show()

# Call the function
categories = ['Healthy', 'Powdery', 'Rust']
train_dir = "/content/drive/MyDrive/Data/Train"  # Update with actual train directory path
visualize_data(train_dir, categories)
```


```
# Build the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(512, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(3, activation='softmax')  # 3 output classes: Healthy, Powdery, Rust
])

# Compile the model
```

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

# Early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=30,
    validation_data=val_generator,
    validation_steps=val_generator.samples // val_generator.batch_size,
    callbacks=[early_stopping])
```

```
42/42 ━━━━━━━━━━━━━━━━━━━━ 180s 4s/step - accuracy: 0.6066 - loss: 0.8073
Epoch 4/30
/usr/local/lib/python3.10/dist-packages/keras/src/callbacks/early_stopping.py:155: UserWarning: Early stopping conditioned on met
  current = self.get_monitor_value(logs)
42/42 ━━━━━━━━━━━━━━━━━━━━ 4s 67ms/step - accuracy: 0.6250 - loss: 0.8258 - val_accuracy: 0.6562 - val_loss: 0.6919
Epoch 5/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 177s 4s/step - accuracy: 0.7023 - loss: 0.6926 - val_accuracy: 0.7857 - val_loss: 0.5279
Epoch 6/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 2s 972us/step - accuracy: 0.8125 - loss: 0.6254
Epoch 7/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 198s 4s/step - accuracy: 0.7743 - loss: 0.5818 - val_accuracy: 0.8125 - val_loss: 0.3794
Epoch 8/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 5s 26ms/step - accuracy: 0.9375 - loss: 0.3558 - val_accuracy: 0.7500 - val_loss: 0.6651
Epoch 9/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 182s 4s/step - accuracy: 0.8607 - loss: 0.4080
Epoch 10/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 4s 65ms/step - accuracy: 0.8125 - loss: 0.3737 - val_accuracy: 0.8125 - val_loss: 0.5225
Epoch 11/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 170s 4s/step - accuracy: 0.8691 - loss: 0.3934 - val_accuracy: 0.8571 - val_loss: 0.2850
Epoch 12/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 3s 657us/step - accuracy: 0.8750 - loss: 0.3020
Epoch 13/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 173s 4s/step - accuracy: 0.8755 - loss: 0.3719 - val_accuracy: 0.8750 - val_loss: 0.4038
Epoch 14/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 3s 16ms/step - accuracy: 0.8438 - loss: 0.5103 - val_accuracy: 0.9286 - val_loss: 0.1879
Epoch 15/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 256s 5s/step - accuracy: 0.8860 - loss: 0.3077
Epoch 16/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 6s 70ms/step - accuracy: 0.8750 - loss: 0.3793 - val_accuracy: 0.8438 - val_loss: 0.3015
Epoch 17/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 173s 4s/step - accuracy: 0.8826 - loss: 0.3083 - val_accuracy: 0.9286 - val_loss: 0.2175
Epoch 18/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 1s 488us/step - accuracy: 0.8571 - loss: 0.1408
Epoch 19/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 176s 4s/step - accuracy: 0.9061 - loss: 0.2651 - val_accuracy: 0.8750 - val_loss: 0.4216
Epoch 20/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 4s 25ms/step - accuracy: 0.9062 - loss: 0.4694 - val_accuracy: 0.9643 - val_loss: 0.1602
Epoch 21/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 199s 4s/step - accuracy: 0.9363 - loss: 0.2194
Epoch 22/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 6s 94ms/step - accuracy: 0.8750 - loss: 0.2305 - val_accuracy: 0.9062 - val_loss: 0.2198
Epoch 23/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 192s 4s/step - accuracy: 0.9259 - loss: 0.2147 - val_accuracy: 0.9286 - val_loss: 0.1996
Epoch 24/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 2s 644us/step - accuracy: 0.9062 - loss: 0.1725
Epoch 25/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 202s 4s/step - accuracy: 0.9140 - loss: 0.2282 - val_accuracy: 0.9375 - val_loss: 0.1441
Epoch 26/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - accuracy: 0.9688 - loss: 0.1143 - val_accuracy: 0.9643 - val_loss: 0.1836
Epoch 27/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 201s 4s/step - accuracy: 0.9029 - loss: 0.2775
Epoch 28/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 5s 87ms/step - accuracy: 0.8438 - loss: 0.3526 - val_accuracy: 0.9375 - val_loss: 0.1014
Epoch 29/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 200s 4s/step - accuracy: 0.9405 - loss: 0.1833 - val_accuracy: 0.9286 - val_loss: 0.1516
Epoch 30/30
42/42 ━━━━━━━━━━━━━━━━━━━━ 2s 672us/step - accuracy: 0.8750 - loss: 0.2601
```

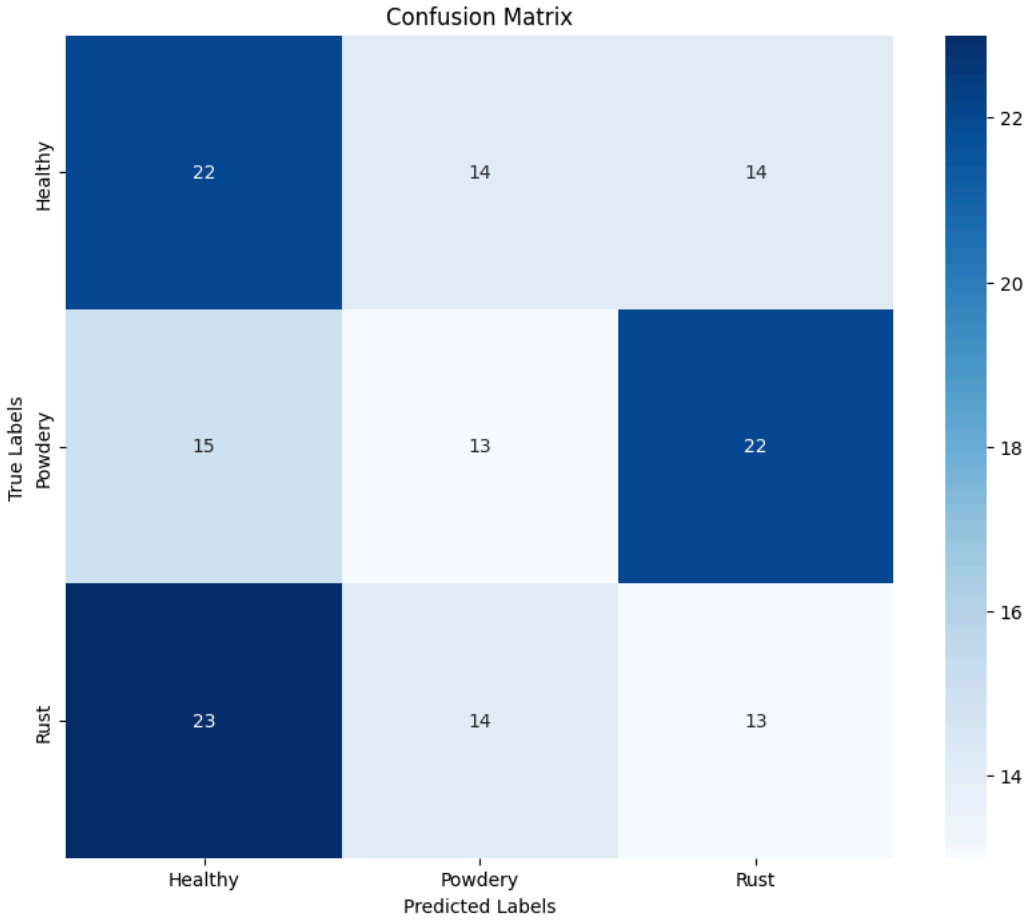
```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_generator, steps=test_generator.samples // test_generator.batch_size)
print(f'Test Accuracy: {test_accuracy*100:.2f}%')
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 26s 8s/step - accuracy: 0.9219 - loss: 0.4540
Test Accuracy: 93.75%
```

```
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
import seaborn as sns

# Ensure the test generator is reset before predictions to avoid missing data
test_generator.reset()

# Make predictions on the test set
predictions = model.predict(test_generator,
                            steps=test_generator.samples // test_generator.batch_size + 1,
                            verbose=1)

# Convert predicted probabilities to class indices
predicted_classes = np.argmax(predictions, axis=1)

# Retrieve true labels and class labels
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

# Confusion Matrix
print("Confusion Matrix:")
cm = confusion_matrix(true_classes, predicted_classes)
print(cm)

# Visualize Confusion Matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels, yticklabels=class_labels)
plt.title("Confusion Matrix")
plt.ylabel("True Labels")
plt.xlabel("Predicted Labels")
plt.show()

# Classification Report
print("\nClassification Report:")
report = classification_report(true_classes, predicted_classes, target_names=class_labels)
print(report)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step
Confusion Matrix:
[[22 14 14]
 [15 13 22]
 [23 14 13]]
```




```
Classification Report:
              precision    recall  f1-score   support

     Healthy       0.37      0.44      0.40        50
     Powdery       0.32      0.26      0.29        50
        Rust       0.27      0.26      0.26        50

    accuracy                           0.32       150
   macro avg       0.32      0.32      0.32       150
weighted avg       0.32      0.32      0.32       150
```

```
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure the test generator is reset before predictions to avoid missing data
test_generator.reset()

# Make predictions on the test set
predictions = model.predict(test_generator,
                            steps=test_generator.samples // test_generator.batch_size + 1,
                            verbose=1)

# Convert predicted probabilities to class indices
predicted_classes = np.argmax(predictions, axis=1)

# Retrieve true labels and class labels
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

# Classification Report
report = classification_report(true_classes, predicted_classes, target_names=class_labels, output_dict=True)

# Convert the classification report to a pandas DataFrame
report_df = pd.DataFrame(report).transpose()

# Display the DataFrame
```

```
print(report_df)

# Visualize the Classification Report
plt.figure(figsize=(10, 6))
sns.heatmap(report_df.iloc[:-1, :-1], annot=True, cmap="YlGnBu", fmt=".2f")
plt.title("Classification Report Heatmap")
plt.ylabel("Classes")
plt.xlabel("Metrics")
plt.show()
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 13s 2s/step
              precision    recall  f1-score     support
Healthy        0.400000  0.480000  0.436364   50.000000
Powdery        0.390244  0.320000  0.351648   50.000000
Rust           0.367347  0.360000  0.363636   50.000000
accuracy       0.386667  0.386667  0.386667    0.386667
macro avg      0.385864  0.386667  0.383883  150.000000
weighted avg   0.385864  0.386667  0.383883  150.000000
```