MINI PROJECT
DEEP LEARNING FOR ELECTRICAL & COMPUTER ENGINEERS
EC9170

Plant Disease Detection using Deep Learning

GROUP: 15

- ATHUPITIYA A.C.D.P. (2020/E/014)
- LIYANAGE L.D.T.N. (2020/E/082)
- MADHUSANKA W.A.D.R. (2020/E/085)

30 JUN 2024

INTRODUCTION

In this report, we explore the application of deep learning models to detect plant diseases using RGB photos of leaves. The dataset comprises two categories: Healthy and Diseased leaves, where Diseased leaves are affected by Apple Scab, Black Rot, or Cedar Apple Rust. Our goal is to develop a model that can accurately classify these images to assist in early disease detection and improve crop management.

We propose and compare two pre-trained deep learning models: VGG16 and ResNet50. By evaluating the performance of these models on a test dataset, we aim to determine the most effective model for this classification task. Our report includes training and validation accuracy over 20 epochs, average accuracies, and test accuracy to provide a comprehensive assessment of each model's performance.

Dataset

The dataset used in this project consists of RGB photos of leaves categorized into two main folders: Healthy and Diseased. The Diseased folder contains images of leaves affected by three specific diseases: Apple Scab, Black Rot, and Cedar Apple Rust. The Healthy folder contains images of green, healthy leaves. This dataset provides a diverse set of examples for both healthy and diseased leaves, which helps in training and evaluating our deep learning models to accurately classify and detect plant diseases.

Dataset Structure

The dataset is structured in directories to facilitate easy access and management. The main directory, ‘plant_disease_detection’, contains three subdirectories: train, validation, and test. Each of these subdirectories is further divided into Healthy and Diseased folders. The train folder contains images used for training the models, the validation folder contains images used for tuning model parameters, and the test folder contains images used to evaluate the final model performance. This directory structure helps in systematically organizing the data for effective model training and evaluation.

Image Preprocessing

Image preprocessing is accomplished using the ‘ImageDataGenerator’ class from TensorFlow's Keras API. This class rescales the pixel values of your images by dividing them by 255 to normalize them to a range of 0 to 1. Additionally, the images are resized to a consistent size of 224x224 pixels, which is the input size required by the VGG16 and ResNet50 models. This preprocessing step ensures that all images fed into the models have a uniform format.

METHODOLOGY

Proposed Models: VGG16, ResNet50

MODEL 1: VGG16

Model Architecture:

1. Input Layer

The input layer expects images of shape (224, 224, 3), which corresponds to 224x224 pixels with 3 color channels (RGB).

2. Convolutional layers

The VGG16 architecture consists of multiple convolutional layers with 3x3 filters and ReLU activation functions. These layers are pre-trained on the ImageNet dataset, and in this project, they are not trainable to leverage the learned features from ImageNet

3. Flatten layer

After the convolutional layers, the output is flattened into a one-dimensional vector. This transformation is necessary to connect the convolutional part of the network to the fully connected layers that follow.

4. Fully connected layers

A Dense layer with 256 units and ReLU activation is added. This layer learns high-level features from the flattened output of the convolutional layers, enabling the model to make more accurate predictions.

5. Output layer

The final layer is a Dense layer with one unit and a sigmoid activation function. This layer is responsible for binary classification, predicting whether a leaf is diseased or healthy.

Training and Evaluation:

The model is compiled with the Adam optimizer (learning rate of 0.0001) and binary cross-entropy loss, suitable for binary classification. Accuracy is used to evaluate performance during training. The 'ImageDataGenerator' class rescales pixel values and splits the dataset into training and validation sets, ensuring appropriately scaled and augmented data. The model is trained for 20 epochs with validation to monitor performance. Post-training, the model's accuracy is evaluated on a separate test dataset to assess its generalization to new data.

MODEL 2: ResNet50

Model Architecture:

1. Input layer

   The ResNet50 model is designed to accept input images of shape (224, 224, 3), which represents 224x224 pixels with three color channels (RGB). This standard input size is required for the pre-trained model.

2. Convolutional layers

   ResNet50 includes multiple convolutional layers organized into residual blocks. These layers are pre-trained on the ImageNet dataset.

3. Flatten layers

   The output from the convolutional layers is flattened into a one-dimensional vector. This flattening step is necessary to connect the convolutional output to the fully connected layers that follow.

4. Fully connected layer

   A Dense layer with 256 units and ReLU activation is added to the model. This fully connected layer processes the high-level features extracted by the convolutional layers, enabling more accurate predictions.

5. Output layers

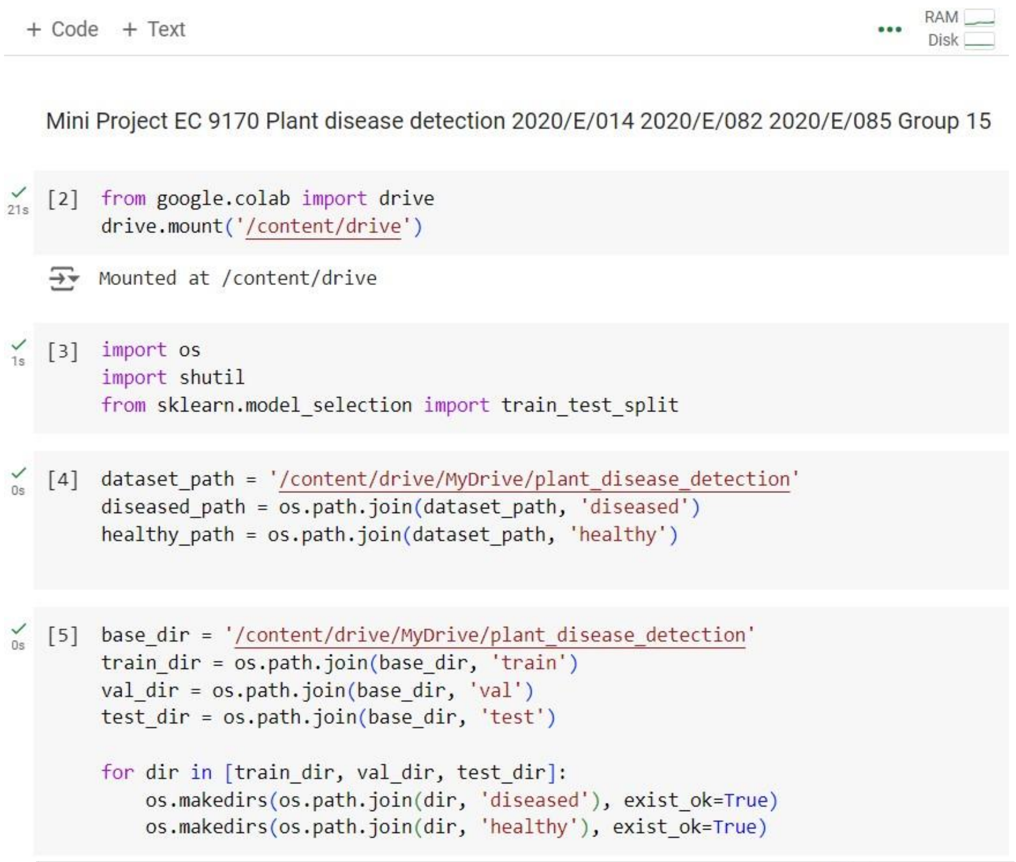
   The final layer is a Dense layer with one unit and a sigmoid activation function. This layer is responsible for binary classification, predicting whether a leaf is diseased or healthy.

Training and Evaluation:

The model is trained for 20 epochs using the training data generator. Validation is performed on the validation data generator to monitor the model's performance on unseen data. After training, the model's accuracy is evaluated using a test data generator to assess its performance on a separate test dataset. This evaluation step ensures that the model generalizes well to new, unseen data.

CODES AND OUTPUTS

1. Mounting google drive and splitting data



Mini Project EC 9170 Plant disease detection 2020/E/014 2020/E/082 2020/E/085 Group 15

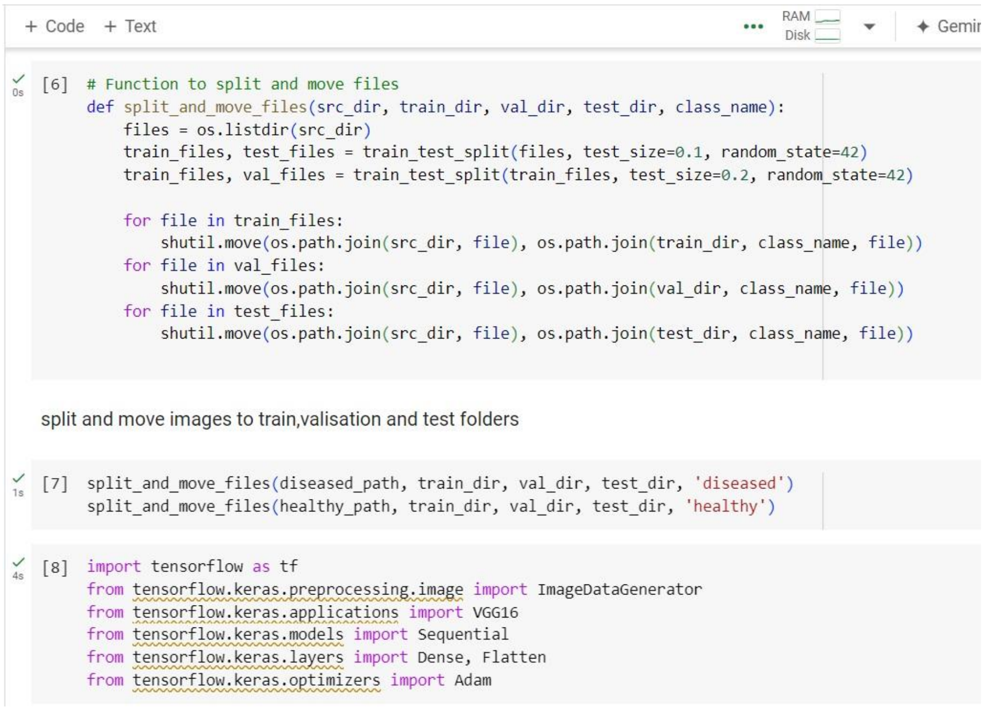
```
[2] from google.colab import drive
    drive.mount('/content/drive')
```

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

```
[3] import os
    import shutil
    from sklearn.model_selection import train_test_split
```

```
[4] dataset_path = '/content/drive/MyDrive/plant_disease_detection'
    diseased_path = os.path.join(dataset_path, 'diseased')
    healthy_path = os.path.join(dataset_path, 'healthy')
```

```
[5] base_dir = '/content/drive/MyDrive/plant_disease_detection'
    train_dir = os.path.join(base_dir, 'train')
    val_dir = os.path.join(base_dir, 'val')
    test_dir = os.path.join(base_dir, 'test')

    for dir in [train_dir, val_dir, test_dir]:
        os.makedirs(os.path.join(dir, 'diseased'), exist_ok=True)
        os.makedirs(os.path.join(dir, 'healthy'), exist_ok=True)
```



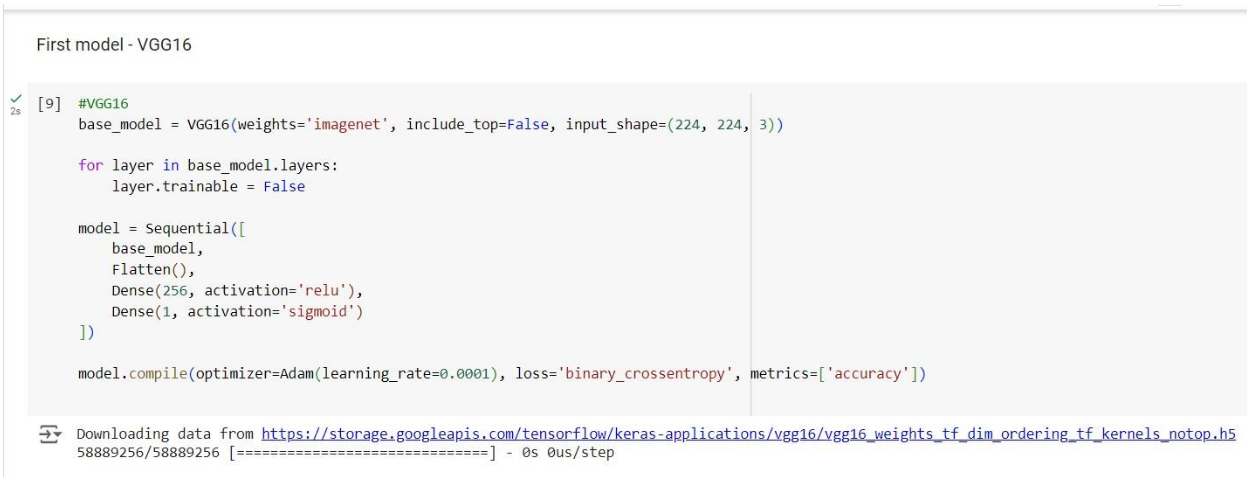
```
[6] # Function to split and move files
    def split_and_move_files(src_dir, train_dir, val_dir, test_dir, class_name):
        files = os.listdir(src_dir)
        train_files, test_files = train_test_split(files, test_size=0.1, random_state=42)
        train_files, val_files = train_test_split(train_files, test_size=0.2, random_state=42)

        for file in train_files:
            shutil.move(os.path.join(src_dir, file), os.path.join(train_dir, class_name, file))
        for file in val_files:
            shutil.move(os.path.join(src_dir, file), os.path.join(val_dir, class_name, file))
        for file in test_files:
            shutil.move(os.path.join(src_dir, file), os.path.join(test_dir, class_name, file))
```

split and move images to train,valisation and test folders

```
[7] split_and_move_files(diseased_path, train_dir, val_dir, test_dir, 'diseased')
    split_and_move_files(healthy_path, train_dir, val_dir, test_dir, 'healthy')
```

```
[8] import tensorflow as tf
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.applications import VGG16
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Flatten
    from tensorflow.keras.optimizers import Adam
```

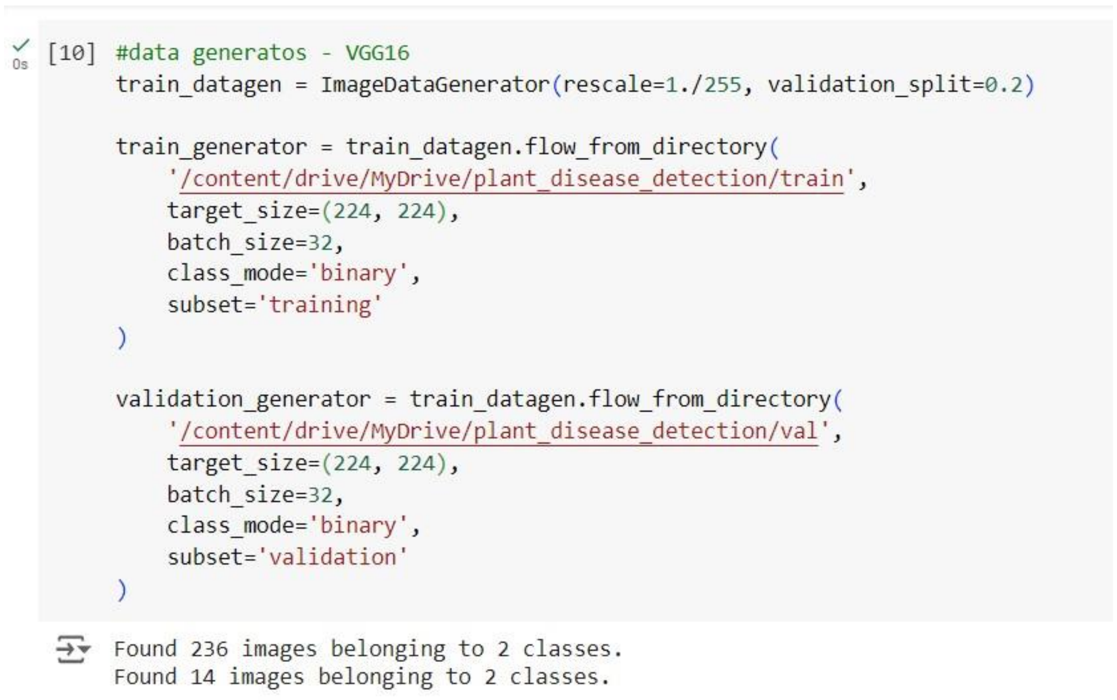
2. First model – VGG16



First model - VGG16

```
[9] #VGG16
    base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

    for layer in base_model.layers:
        layer.trainable = False

    model = Sequential([
        base_model,
        Flatten(),
        Dense(256, activation='relu'),
        Dense(1, activation='sigmoid')
    ])

    model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
```

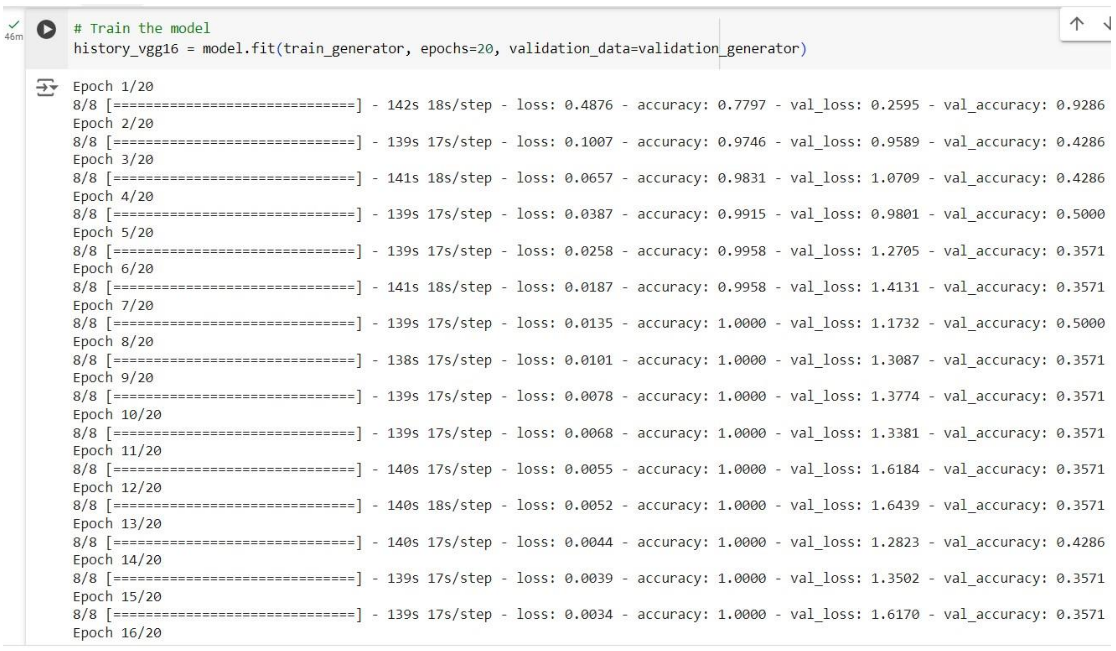
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
```



```
[10] #data generatos - VGG16
     train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

     train_generator = train_datagen.flow_from_directory(
         '/content/drive/MyDrive/plant_disease_detection/train',
         target_size=(224, 224),
         batch_size=32,
         class_mode='binary',
         subset='training'
     )

     validation_generator = train_datagen.flow_from_directory(
         '/content/drive/MyDrive/plant_disease_detection/val',
         target_size=(224, 224),
         batch_size=32,
         class_mode='binary',
         subset='validation'
     )
```

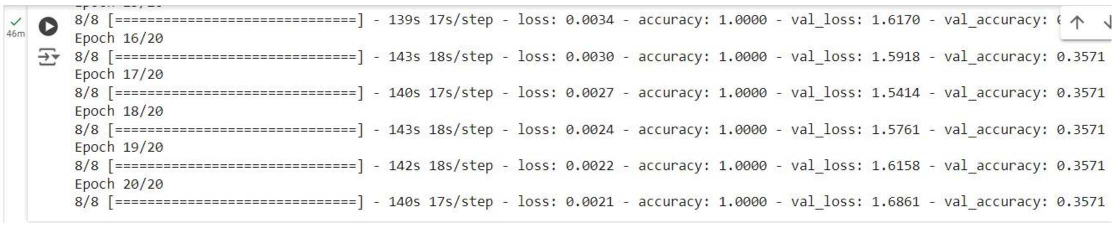
```
Found 236 images belonging to 2 classes.
Found 14 images belonging to 2 classes.
```

Training the model – VGG16



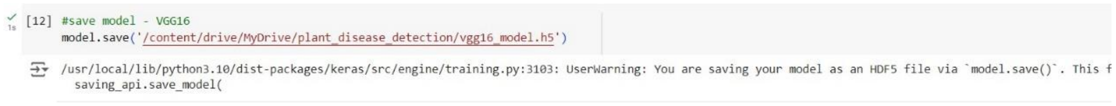
```
# Train the model
history_vgg16 = model.fit(train_generator, epochs=20, validation_data=validation_generator)
```

```
Epoch 1/20
8/8 [==============================] - 142s 18s/step - loss: 0.4876 - accuracy: 0.7797 - val_loss: 0.2595 - val_accuracy: 0.9286
Epoch 2/20
8/8 [==============================] - 139s 17s/step - loss: 0.1007 - accuracy: 0.9746 - val_loss: 0.9589 - val_accuracy: 0.4286
Epoch 3/20
8/8 [==============================] - 141s 18s/step - loss: 0.0657 - accuracy: 0.9831 - val_loss: 1.0709 - val_accuracy: 0.4286
Epoch 4/20
8/8 [==============================] - 139s 17s/step - loss: 0.0387 - accuracy: 0.9915 - val_loss: 0.9801 - val_accuracy: 0.5000
Epoch 5/20
8/8 [==============================] - 139s 17s/step - loss: 0.0258 - accuracy: 0.9958 - val_loss: 1.2705 - val_accuracy: 0.3571
Epoch 6/20
8/8 [==============================] - 141s 18s/step - loss: 0.0187 - accuracy: 0.9958 - val_loss: 1.4131 - val_accuracy: 0.3571
Epoch 7/20
8/8 [==============================] - 139s 17s/step - loss: 0.0135 - accuracy: 1.0000 - val_loss: 1.1732 - val_accuracy: 0.5000
Epoch 8/20
8/8 [==============================] - 138s 17s/step - loss: 0.0101 - accuracy: 1.0000 - val_loss: 1.3087 - val_accuracy: 0.3571
Epoch 9/20
8/8 [==============================] - 139s 17s/step - loss: 0.0078 - accuracy: 1.0000 - val_loss: 1.3774 - val_accuracy: 0.3571
Epoch 10/20
8/8 [==============================] - 139s 17s/step - loss: 0.0068 - accuracy: 1.0000 - val_loss: 1.3381 - val_accuracy: 0.3571
Epoch 11/20
8/8 [==============================] - 140s 17s/step - loss: 0.0055 - accuracy: 1.0000 - val_loss: 1.6184 - val_accuracy: 0.3571
Epoch 12/20
8/8 [==============================] - 140s 18s/step - loss: 0.0052 - accuracy: 1.0000 - val_loss: 1.6439 - val_accuracy: 0.3571
Epoch 13/20
8/8 [==============================] - 140s 17s/step - loss: 0.0044 - accuracy: 1.0000 - val_loss: 1.2823 - val_accuracy: 0.4286
Epoch 14/20
8/8 [==============================] - 139s 17s/step - loss: 0.0039 - accuracy: 1.0000 - val_loss: 1.3502 - val_accuracy: 0.3571
Epoch 15/20
8/8 [==============================] - 139s 17s/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 1.6170 - val_accuracy: 0.3571
Epoch 16/20
```



```
8/8 [==============================] - 139s 17s/step - loss: 0.0034 - accuracy: 1.0000 - val_loss: 1.6170 - val_accuracy: (
Epoch 16/20
8/8 [==============================] - 143s 18s/step - loss: 0.0030 - accuracy: 1.0000 - val_loss: 1.5918 - val_accuracy: 0.3571
Epoch 17/20
8/8 [==============================] - 140s 17s/step - loss: 0.0027 - accuracy: 1.0000 - val_loss: 1.5414 - val_accuracy: 0.3571
Epoch 18/20
8/8 [==============================] - 143s 18s/step - loss: 0.0024 - accuracy: 1.0000 - val_loss: 1.5761 - val_accuracy: 0.3571
Epoch 19/20
8/8 [==============================] - 142s 18s/step - loss: 0.0022 - accuracy: 1.0000 - val_loss: 1.6158 - val_accuracy: 0.3571
Epoch 20/20
8/8 [==============================] - 140s 17s/step - loss: 0.0021 - accuracy: 1.0000 - val_loss: 1.6861 - val_accuracy: 0.3571
```

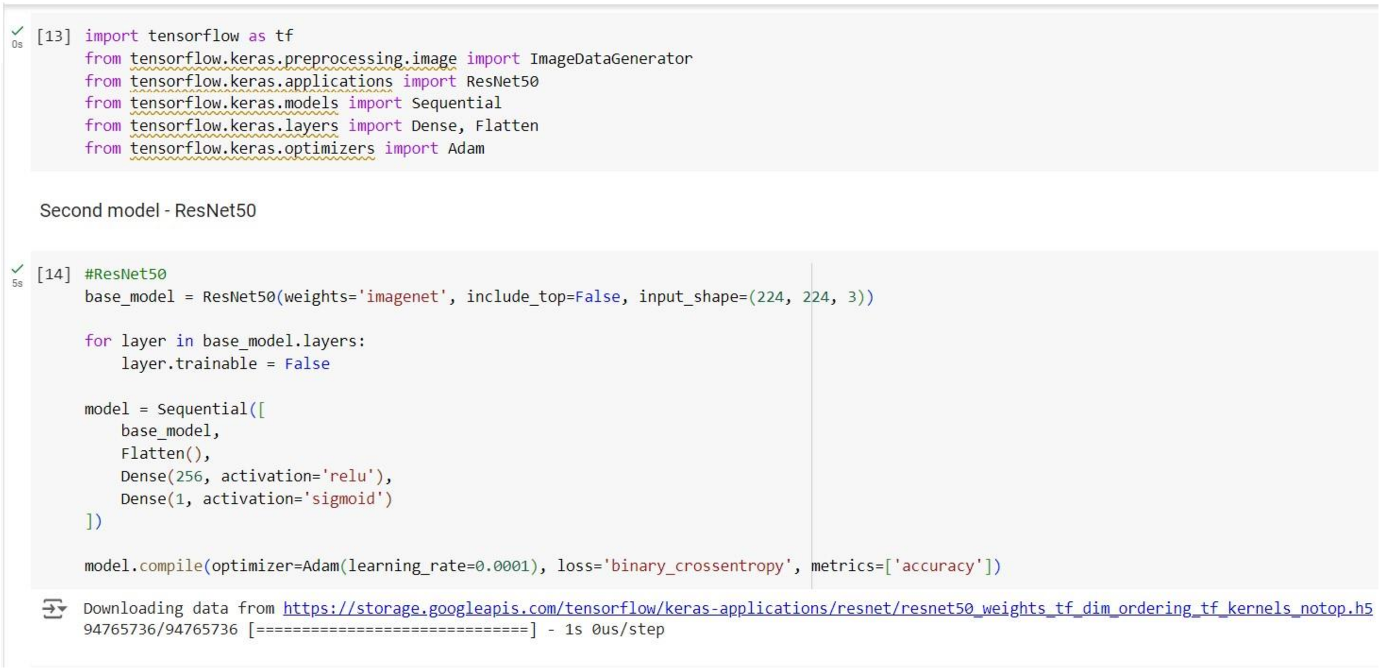
Saving the model – VGG16



```
[12] #save model - VGG16
     model.save('/content/drive/MyDrive/plant_disease_detection/vgg16_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This f
  saving_api.save_model(
```

3. Second model – ResNet50

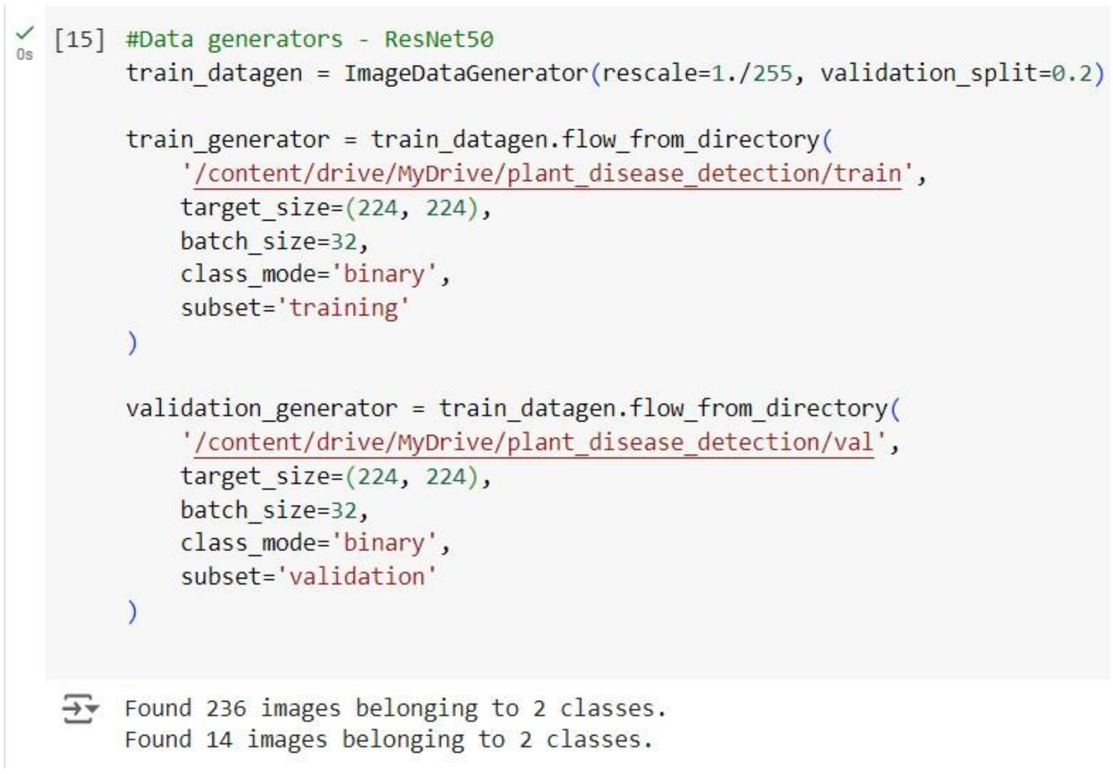


[13]
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
```

Second model - ResNet50

[14]
```
#ResNet50
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

for layer in base_model.layers:
    layer.trainable = False

model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
```

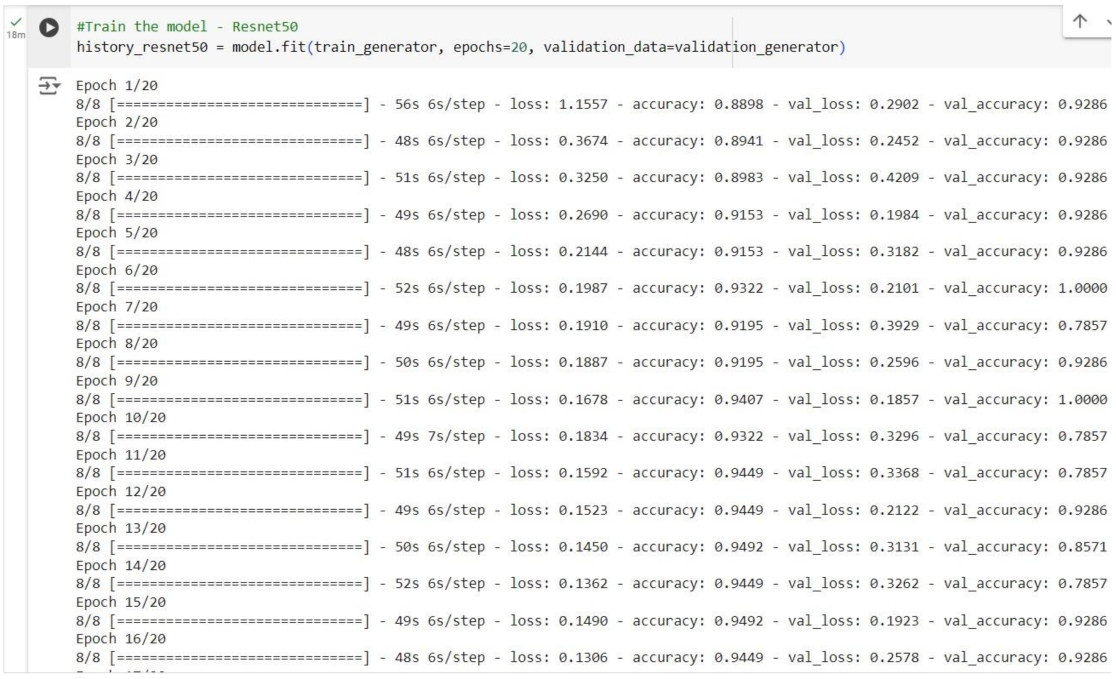
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 1s 0us/step
```



[15]
```
#Data generators - ResNet50
train_datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

train_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/plant_disease_detection/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/plant_disease_detection/val',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)
```

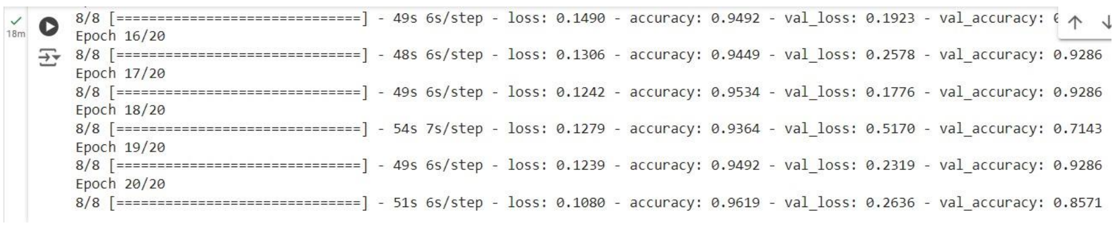
```
Found 236 images belonging to 2 classes.
Found 14 images belonging to 2 classes.
```

Training the model – ResNet50



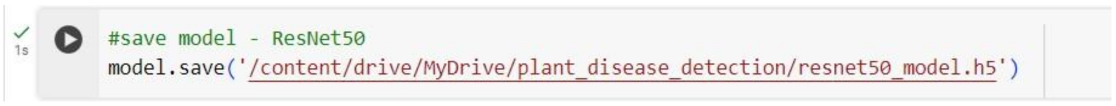
```
#Train the model - Resnet50
history_resnet50 = model.fit(train_generator, epochs=20, validation_data=validation_generator)
```

```
Epoch 1/20
8/8 [==============================] - 56s 6s/step - loss: 1.1557 - accuracy: 0.8898 - val_loss: 0.2902 - val_accuracy: 0.9286
Epoch 2/20
8/8 [==============================] - 48s 6s/step - loss: 0.3674 - accuracy: 0.8941 - val_loss: 0.2452 - val_accuracy: 0.9286
Epoch 3/20
8/8 [==============================] - 51s 6s/step - loss: 0.3250 - accuracy: 0.8983 - val_loss: 0.4209 - val_accuracy: 0.9286
Epoch 4/20
8/8 [==============================] - 49s 6s/step - loss: 0.2690 - accuracy: 0.9153 - val_loss: 0.1984 - val_accuracy: 0.9286
Epoch 5/20
8/8 [==============================] - 48s 6s/step - loss: 0.2144 - accuracy: 0.9153 - val_loss: 0.3182 - val_accuracy: 0.9286
Epoch 6/20
8/8 [==============================] - 52s 6s/step - loss: 0.1987 - accuracy: 0.9322 - val_loss: 0.2101 - val_accuracy: 1.0000
Epoch 7/20
8/8 [==============================] - 49s 6s/step - loss: 0.1910 - accuracy: 0.9195 - val_loss: 0.3929 - val_accuracy: 0.7857
Epoch 8/20
8/8 [==============================] - 50s 6s/step - loss: 0.1887 - accuracy: 0.9195 - val_loss: 0.2596 - val_accuracy: 0.9286
Epoch 9/20
8/8 [==============================] - 51s 6s/step - loss: 0.1678 - accuracy: 0.9407 - val_loss: 0.1857 - val_accuracy: 1.0000
Epoch 10/20
8/8 [==============================] - 49s 7s/step - loss: 0.1834 - accuracy: 0.9322 - val_loss: 0.3296 - val_accuracy: 0.7857
Epoch 11/20
8/8 [==============================] - 51s 6s/step - loss: 0.1592 - accuracy: 0.9449 - val_loss: 0.3368 - val_accuracy: 0.7857
Epoch 12/20
8/8 [==============================] - 49s 6s/step - loss: 0.1523 - accuracy: 0.9449 - val_loss: 0.2122 - val_accuracy: 0.9286
Epoch 13/20
8/8 [==============================] - 50s 6s/step - loss: 0.1450 - accuracy: 0.9492 - val_loss: 0.3131 - val_accuracy: 0.8571
Epoch 14/20
8/8 [==============================] - 52s 6s/step - loss: 0.1362 - accuracy: 0.9449 - val_loss: 0.3262 - val_accuracy: 0.7857
Epoch 15/20
8/8 [==============================] - 49s 6s/step - loss: 0.1490 - accuracy: 0.9492 - val_loss: 0.1923 - val_accuracy: 0.9286
Epoch 16/20
8/8 [==============================] - 48s 6s/step - loss: 0.1306 - accuracy: 0.9449 - val_loss: 0.2578 - val_accuracy: 0.9286
```



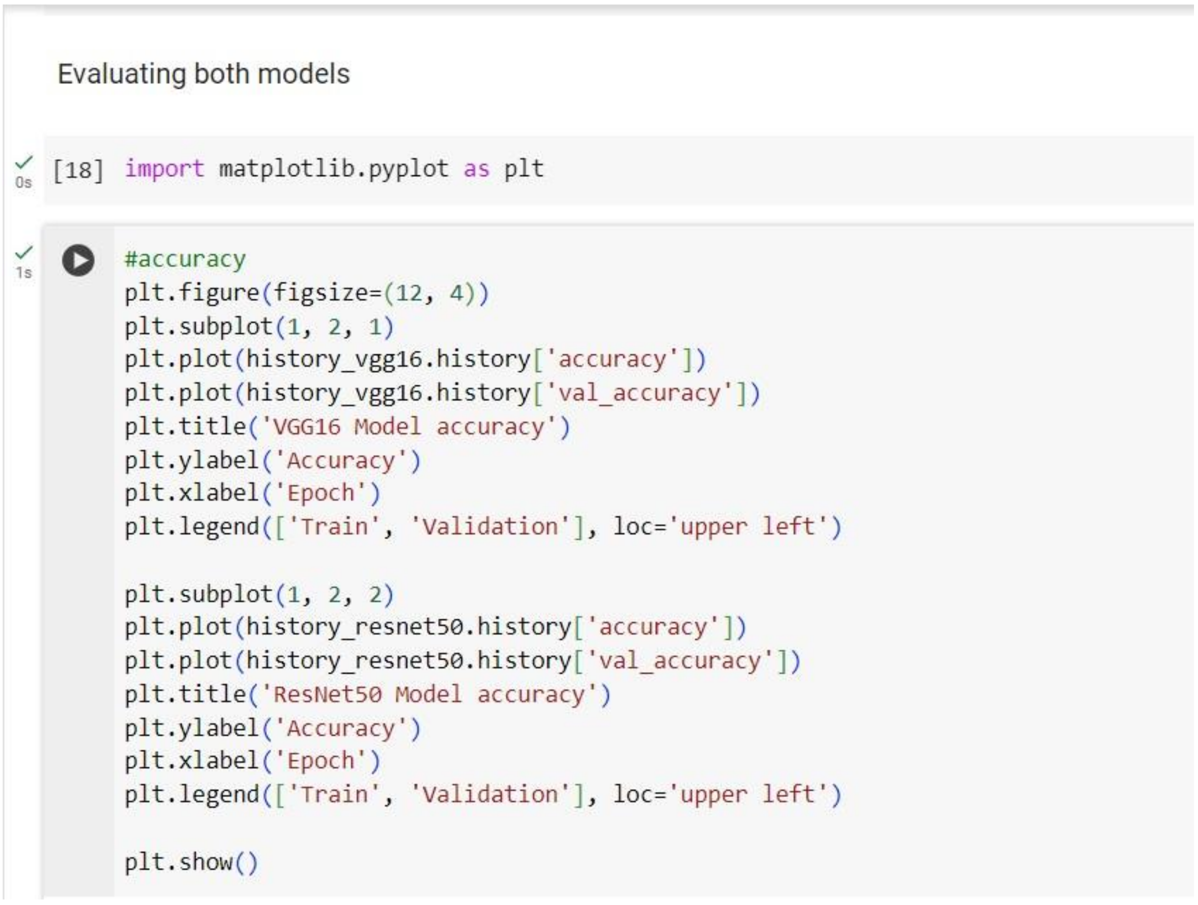
```
8/8 [==============================] - 49s 6s/step - loss: 0.1490 - accuracy: 0.9492 - val_loss: 0.1923 - val_accuracy:
Epoch 16/20
8/8 [==============================] - 48s 6s/step - loss: 0.1306 - accuracy: 0.9449 - val_loss: 0.2578 - val_accuracy: 0.9286
Epoch 17/20
8/8 [==============================] - 49s 6s/step - loss: 0.1242 - accuracy: 0.9534 - val_loss: 0.1776 - val_accuracy: 0.9286
Epoch 18/20
8/8 [==============================] - 54s 7s/step - loss: 0.1279 - accuracy: 0.9364 - val_loss: 0.5170 - val_accuracy: 0.7143
Epoch 19/20
8/8 [==============================] - 49s 6s/step - loss: 0.1239 - accuracy: 0.9492 - val_loss: 0.2319 - val_accuracy: 0.9286
Epoch 20/20
8/8 [==============================] - 51s 6s/step - loss: 0.1080 - accuracy: 0.9619 - val_loss: 0.2636 - val_accuracy: 0.8571
```

Saving the model – ResNet50



```
#save model - ResNet50
model.save('/content/drive/MyDrive/plant_disease_detection/resnet50_model.h5')
```

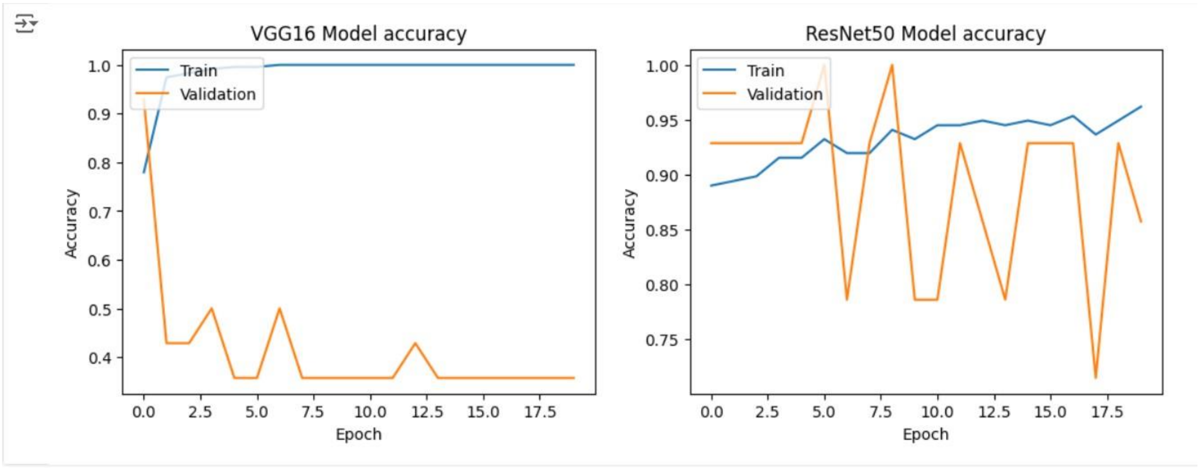
4. Evaluation and comparison between two models.

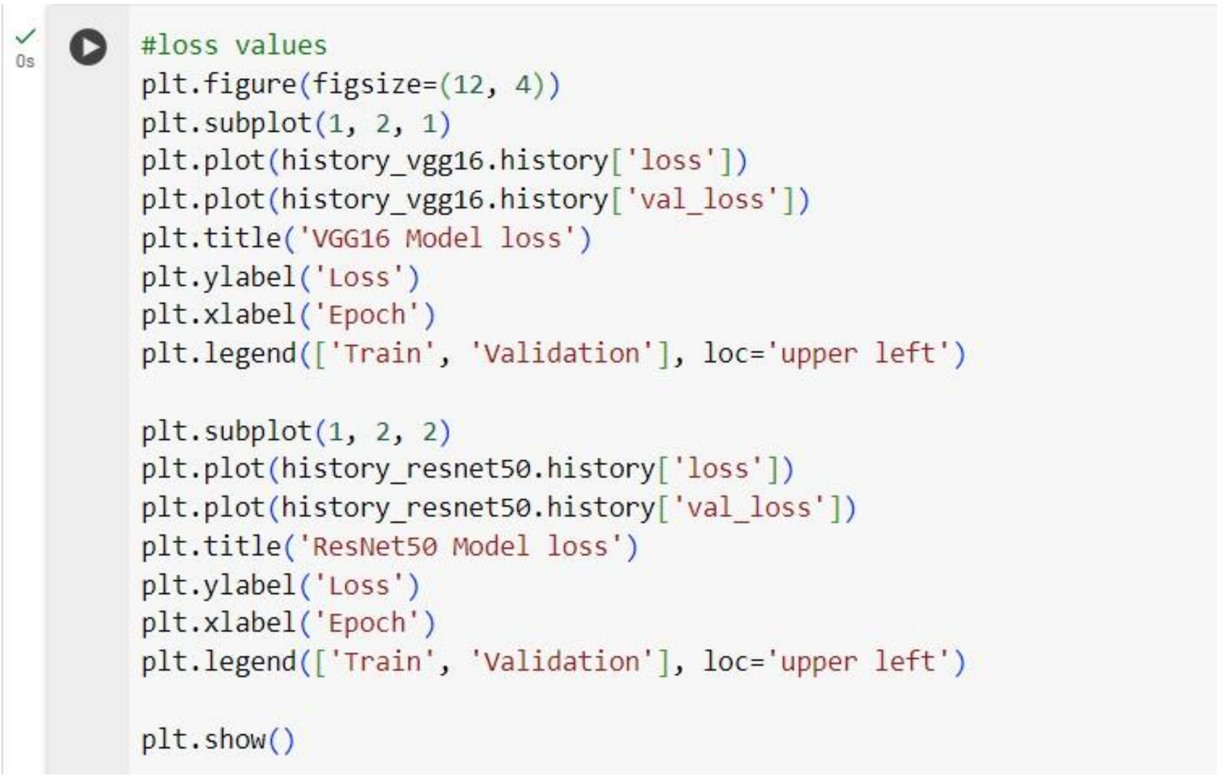


Evaluating both models

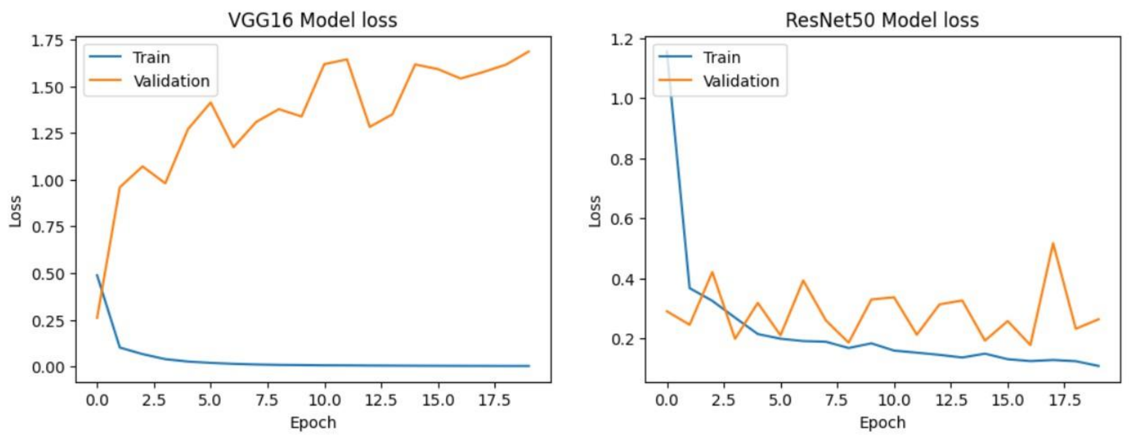
```
[18] import matplotlib.pyplot as plt
```

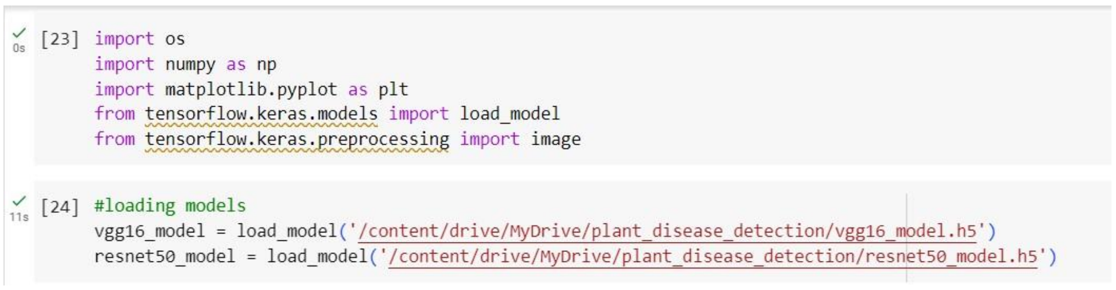
```
#accuracy
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_vgg16.history['accuracy'])
plt.plot(history_vgg16.history['val_accuracy'])
plt.title('VGG16 Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history_resnet50.history['accuracy'])
plt.plot(history_resnet50.history['val_accuracy'])
plt.title('ResNet50 Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()
```





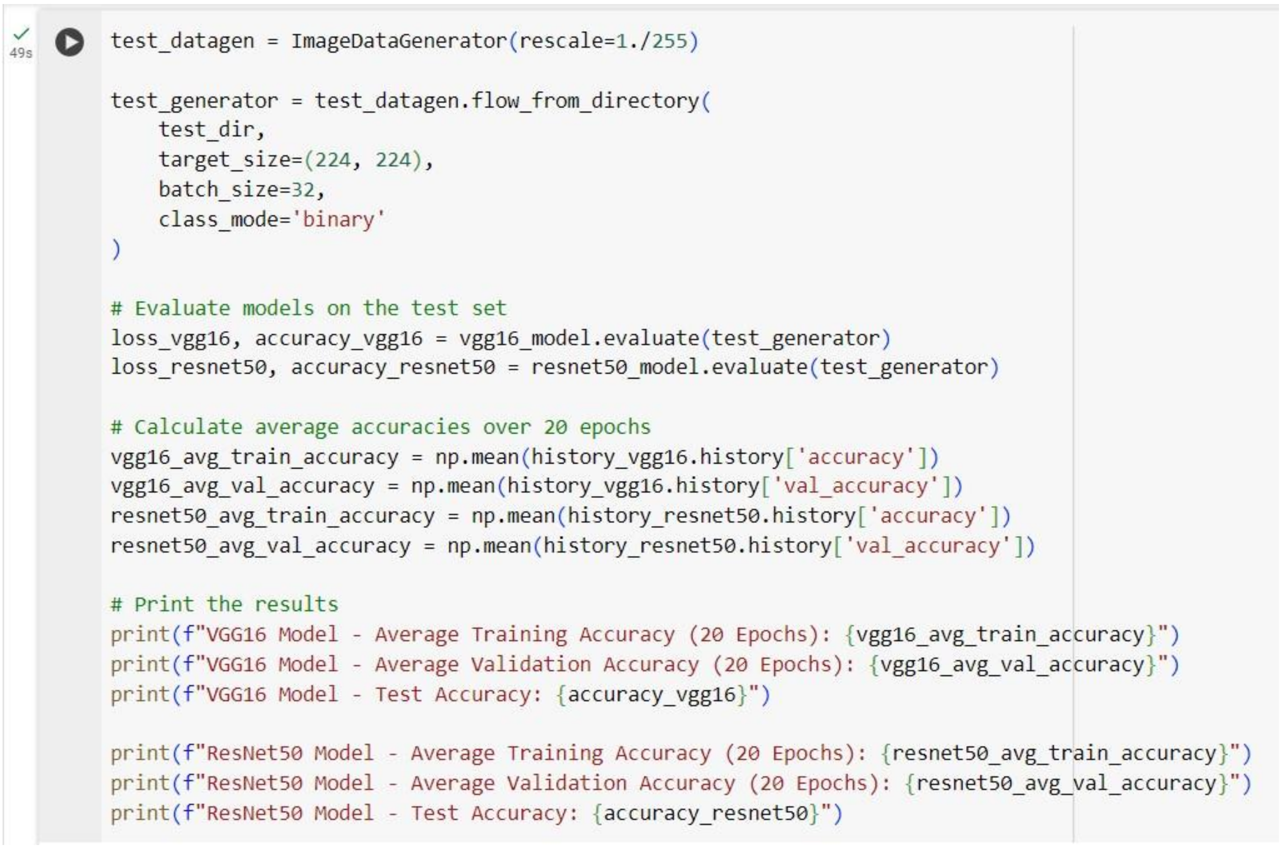
```
#loss values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history_vgg16.history['loss'])
plt.plot(history_vgg16.history['val_loss'])
plt.title('VGG16 Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.subplot(1, 2, 2)
plt.plot(history_resnet50.history['loss'])
plt.plot(history_resnet50.history['val_loss'])
plt.title('ResNet50 Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()
```



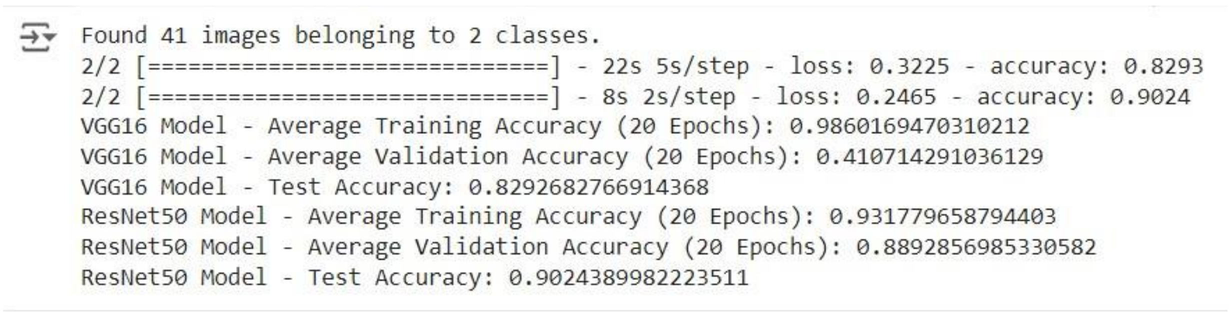


```
[23] import os
     import numpy as np
     import matplotlib.pyplot as plt
     from tensorflow.keras.models import load_model
     from tensorflow.keras.preprocessing import image
```

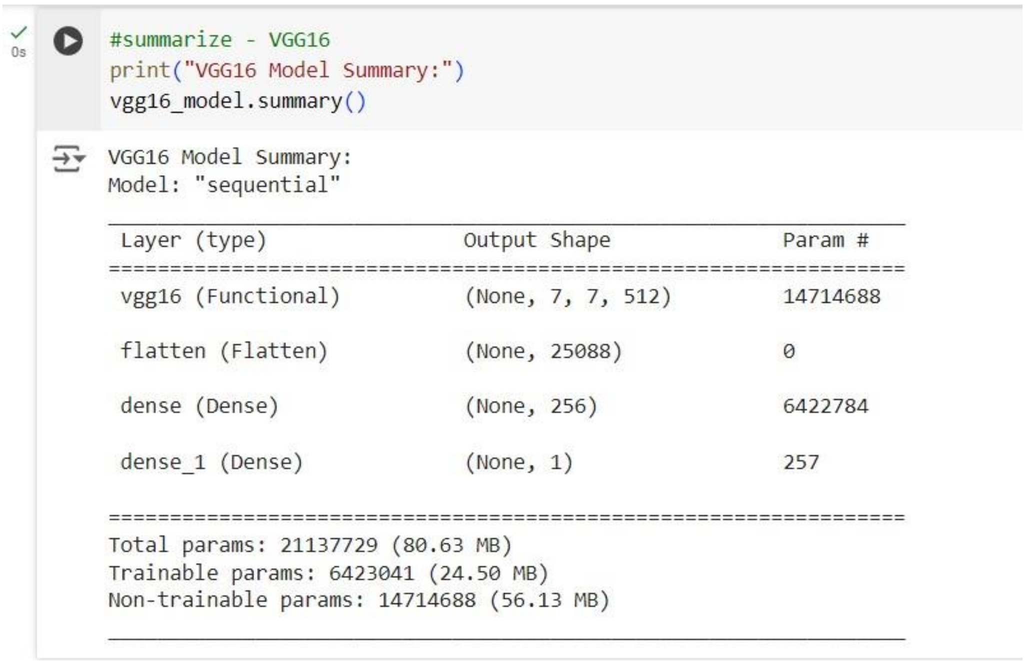
```
[24] #loading models
     vgg16_model = load_model('/content/drive/MyDrive/plant_disease_detection/vgg16_model.h5')
     resnet50_model = load_model('/content/drive/MyDrive/plant_disease_detection/resnet50_model.h5')
```



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

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary'
)

# Evaluate models on the test set
loss_vgg16, accuracy_vgg16 = vgg16_model.evaluate(test_generator)
loss_resnet50, accuracy_resnet50 = resnet50_model.evaluate(test_generator)

# Calculate average accuracies over 20 epochs
vgg16_avg_train_accuracy = np.mean(history_vgg16.history['accuracy'])
vgg16_avg_val_accuracy = np.mean(history_vgg16.history['val_accuracy'])
resnet50_avg_train_accuracy = np.mean(history_resnet50.history['accuracy'])
resnet50_avg_val_accuracy = np.mean(history_resnet50.history['val_accuracy'])

# Print the results
print(f"VGG16 Model - Average Training Accuracy (20 Epochs): {vgg16_avg_train_accuracy}")
print(f"VGG16 Model - Average Validation Accuracy (20 Epochs): {vgg16_avg_val_accuracy}")
print(f"VGG16 Model - Test Accuracy: {accuracy_vgg16}")

print(f"ResNet50 Model - Average Training Accuracy (20 Epochs): {resnet50_avg_train_accuracy}")
print(f"ResNet50 Model - Average Validation Accuracy (20 Epochs): {resnet50_avg_val_accuracy}")
print(f"ResNet50 Model - Test Accuracy: {accuracy_resnet50}")
```

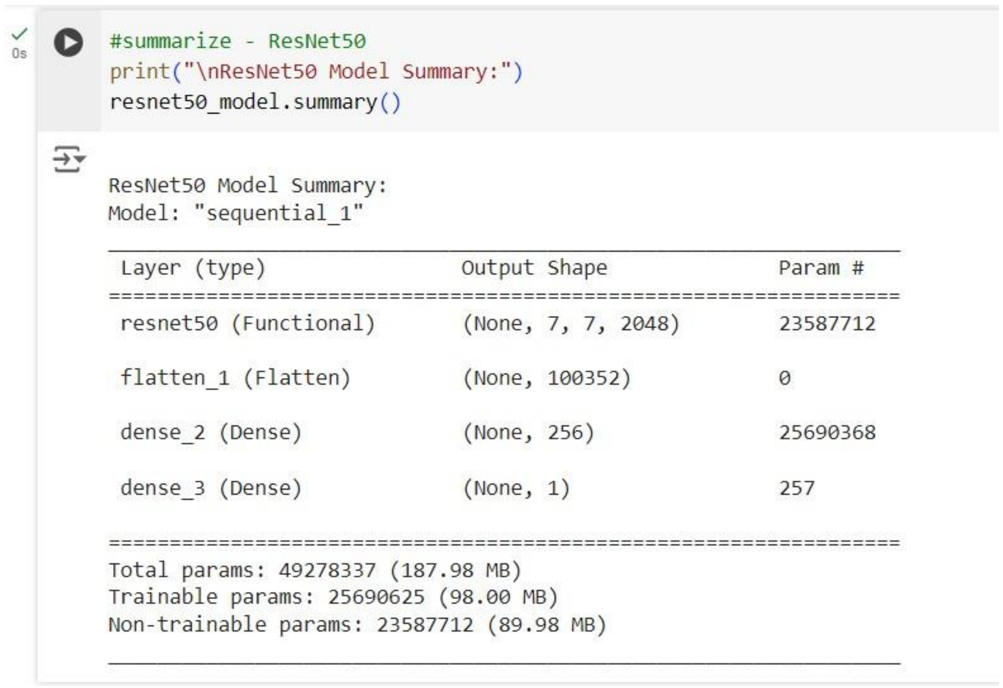


```
Found 41 images belonging to 2 classes.
2/2 [==============================] - 22s 5s/step - loss: 0.3225 - accuracy: 0.8293
2/2 [==============================] - 8s 2s/step - loss: 0.2465 - accuracy: 0.9024
VGG16 Model - Average Training Accuracy (20 Epochs): 0.9860169470310212
VGG16 Model - Average Validation Accuracy (20 Epochs): 0.410714291036129
VGG16 Model - Test Accuracy: 0.8292682766914368
ResNet50 Model - Average Training Accuracy (20 Epochs): 0.931779658794403
ResNet50 Model - Average Validation Accuracy (20 Epochs): 0.8892856985330582
ResNet50 Model - Test Accuracy: 0.9024389982223511
```



```
#summarize - VGG16
print("VGG16 Model Summary:")
vgg16_model.summary()
```

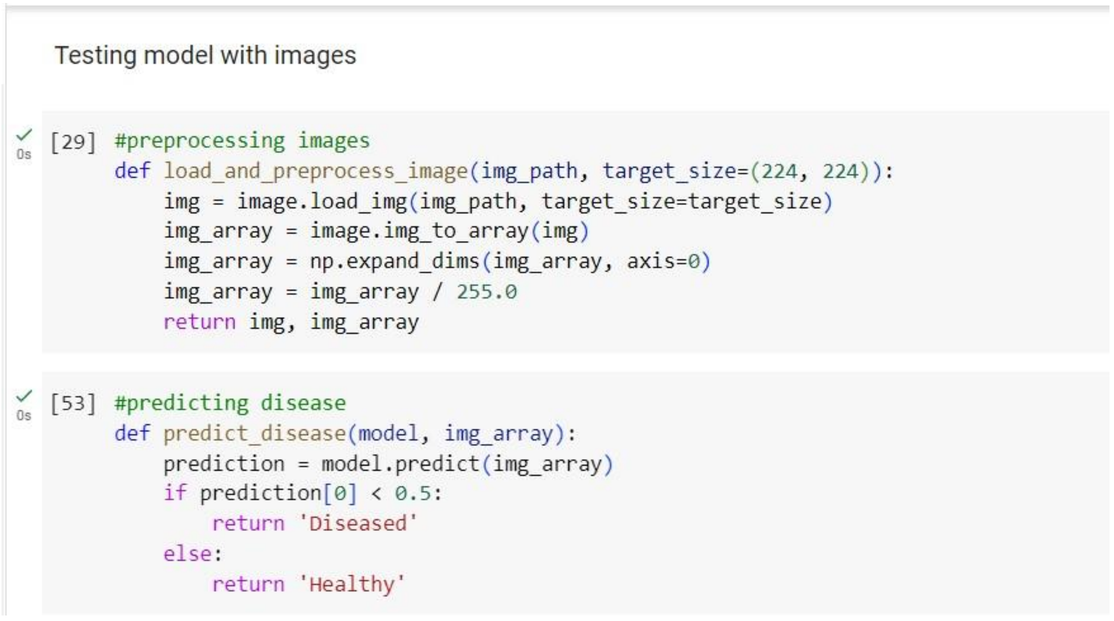
```
VGG16 Model Summary:
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 256)               6422784

 dense_1 (Dense)             (None, 1)                 257

=================================================================
Total params: 21137729 (80.63 MB)
Trainable params: 6423041 (24.50 MB)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```



```
#summarize - ResNet50
print("\nResNet50 Model Summary:")
resnet50_model.summary()
```

```
ResNet50 Model Summary:
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 flatten_1 (Flatten)         (None, 100352)            0

 dense_2 (Dense)             (None, 256)               25690368

 dense_3 (Dense)             (None, 1)                 257

=================================================================
Total params: 49278337 (187.98 MB)
Trainable params: 25690625 (98.00 MB)
Non-trainable params: 23587712 (89.98 MB)
_________________________________________________________________
```

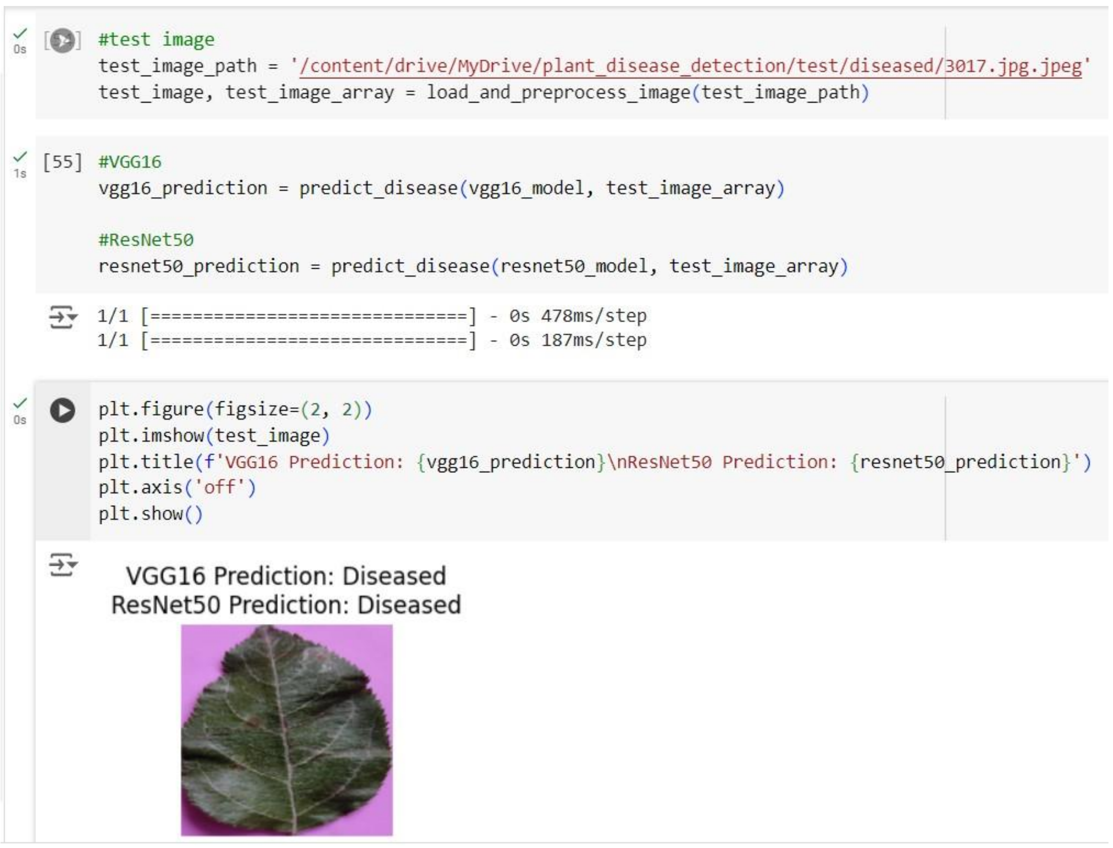
5. Testing the model



Testing model with images

```
[29] #preprocessing images
     def load_and_preprocess_image(img_path, target_size=(224, 224)):
         img = image.load_img(img_path, target_size=target_size)
         img_array = image.img_to_array(img)
         img_array = np.expand_dims(img_array, axis=0)
         img_array = img_array / 255.0
         return img, img_array
```

```
[53] #predicting disease
     def predict_disease(model, img_array):
         prediction = model.predict(img_array)
         if prediction[0] < 0.5:
             return 'Diseased'
         else:
             return 'Healthy'
```



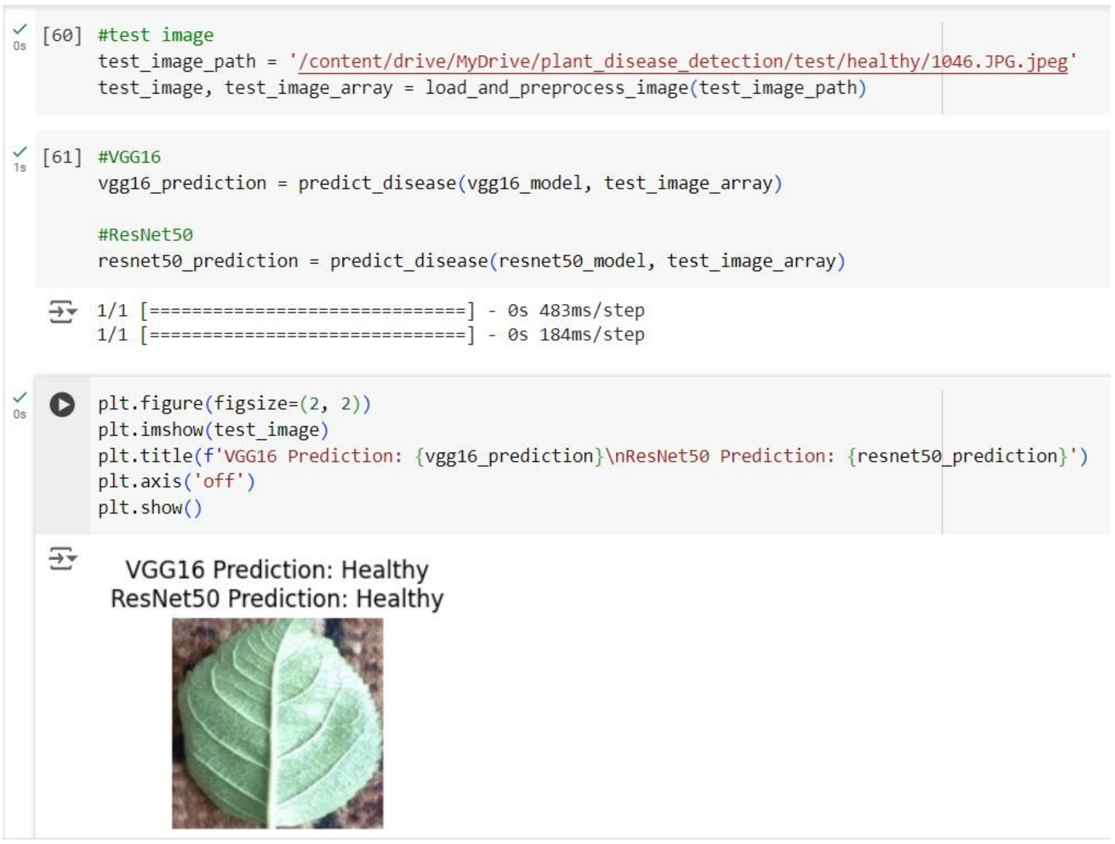
```
#test image
test_image_path = '/content/drive/MyDrive/plant_disease_detection/test/diseased/3017.jpg.jpeg'
test_image, test_image_array = load_and_preprocess_image(test_image_path)
```

```
[55] #VGG16
vgg16_prediction = predict_disease(vgg16_model, test_image_array)

#ResNet50
resnet50_prediction = predict_disease(resnet50_model, test_image_array)
```

```
1/1 [==============================] - 0s 478ms/step
1/1 [==============================] - 0s 187ms/step
```

```
plt.figure(figsize=(2, 2))
plt.imshow(test_image)
plt.title(f'VGG16 Prediction: {vgg16_prediction}\nResNet50 Prediction: {resnet50_prediction}')
plt.axis('off')
plt.show()
```

VGG16 Prediction: Diseased
ResNet50 Prediction: Diseased



```
[60] #test image
test_image_path = '/content/drive/MyDrive/plant_disease_detection/test/healthy/1046.JPG.jpeg'
test_image, test_image_array = load_and_preprocess_image(test_image_path)
```

```
[61] #VGG16
vgg16_prediction = predict_disease(vgg16_model, test_image_array)

#ResNet50
resnet50_prediction = predict_disease(resnet50_model, test_image_array)
```

```
1/1 [==============================] - 0s 483ms/step
1/1 [==============================] - 0s 184ms/step
```

```
plt.figure(figsize=(2, 2))
plt.imshow(test_image)
plt.title(f'VGG16 Prediction: {vgg16_prediction}\nResNet50 Prediction: {resnet50_prediction}')
plt.axis('off')
plt.show()
```

VGG16 Prediction: Healthy
ResNet50 Prediction: Healthy

COMPARISON BETWEEN TWO MODELS

Metric	VGG16	ResNet50
Average Training Accuracy (20 Epochs)	98.61%	93.18%
Average Validation Accuracy (20 Epochs)	41.07%	88.93%
Test Accuracy	82.93%	90.24%
Training Time	Longer	Shorter
Model Complexity	Simpler	More Complex
Resource Requirements	Lower	Higher

1. Training Accuracy:

The VGG16 model achieves a very high average training accuracy of 98.61%, indicating that it performs exceptionally well on the training data.

The ResNet50 model also shows strong performance on the training data with an average accuracy of 93.18%, although slightly lower than VGG16.

2. Validation Accuracy:

There is a significant drop in the VGG16 model's validation accuracy, averaging 41.07%, which suggests overfitting. The model performs well on training data but poorly on validation data.

The ResNet50 model maintains a high average validation accuracy of 88.93%, indicating better generalization to unseen data compared to VGG16.

3. Test Accuracy:

The VGG16 model achieves a test accuracy of 82.93%, which is higher than its validation accuracy but still lower than the ResNet50's test accuracy.

The ResNet50 model achieves a test accuracy of 90.24%, confirming its robustness and better generalization capability compared to the VGG16 model.

4. Training Time:

VGG16 takes longer per training step compared to ResNet50.

5. Model Complexity:

VGG16 is simpler with fewer layers, while ResNet50 is more complex with a deeper architecture.

6. Resource Requirements:

VGG16 requires fewer computational resources, whereas ResNet50 demands more due to its increased number of layers and complexity.

PROPOSAL OF THE BEST MODEL

Best model – ResNet50

Overfitting in VGG16:

The VGG16 model shows signs of overfitting, as evidenced by the large discrepancy between training and validation accuracies. Despite its high training accuracy, the model struggles to generalize to validation and test datasets.

Better Generalization in ResNet50:

The ResNet50 model exhibits better generalization with consistent performance across training, validation, and test datasets. This model is more reliable for real-world applications due to its higher test accuracy and balanced performance.

From the evaluation and comparison, we propose that **ResNet50** would be the best choice for plant disease detection.

CONCLUSION

Based on the detailed evaluation, observations, and field experience, **ResNet50** is proposed as the best model for plant disease detection. Its balanced performance, resource efficiency, and better generalization make it a more suitable and practical choice for real-world applications. While the VGG16 model leveraged pre-trained knowledge, its inability to generalize well to validation and test sets limits its practical utility in this specific task.