Tomato Disease Classification

Import all the Dependencies

```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML

print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.9.2
```

Perform datasplit on dataset

Download dataset from https://www.kaggle.com/datasets/arjuntejaswi/plant-village/data and extract here

Used splitfolders tool to split dataset into training, validation and test directories.

$ pip install split-folders

$ splitfolders --ratio 0.8 0.1 0.1 --output ./datasplit ./PlantVillage/

```
IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS = 3
EPOCHS = 50

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Load dataset with ImageDataGenerator for augmentation
datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    horizontal_flip=True
)

train_generator = datagen.flow_from_directory(
    'datasplit/train',
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE,
    class_mode='sparse',
)
```

```
Found 12804 images belonging to 10 classes.
```

```
train_generator.class_indices
```

```
{'Tomato_Bacterial_spot': 0,
 'Tomato_Early_blight': 1,
 'Tomato_Late_blight': 2,
 'Tomato_Leaf_Mold': 3,
 'Tomato_Septoria_leaf_spot': 4,
 'Tomato_Spider_mites_Two_spotted_spider_mite': 5,
 'Tomato__Target_Spot': 6,
 'Tomato__Tomato_YellowLeaf__Curl_Virus': 7,
 'Tomato__Tomato_mosaic_virus': 8,
 'Tomato_healthy': 9}
```

```
class_names = list(train_generator.class_indices.keys())
class_names
```

```
['Tomato_Bacterial_spot',
 'Tomato_Early_blight',
 'Tomato_Late_blight',
 'Tomato_Leaf_Mold',
 'Tomato_Septoria_leaf_spot',
 'Tomato_Spider_mites_Two_spotted_spider_mite',
 'Tomato__Target_Spot',
 'Tomato__Tomato_YellowLeaf__Curl_Virus',
 'Tomato__Tomato_mosaic_virus',
 'Tomato_healthy']
```

```
count=0
for image_batch, label_batch in train_generator:
#     print(label_batch)
    print(image_batch[0])
    break
#     count+=1
#     if count>2:
#         break
```

```
[[[0.593534   0.53471047 0.52294576]
  [0.6039216  0.54509807 0.53333336]
  [0.59777963 0.5389561  0.5271914 ]
  ...
  [0.748015   0.68919146 0.67742676]
  [0.7556561  0.6967761  0.6850302 ]
  [0.7274847  0.6672841  0.65597844]]

 [[0.59221345 0.5333899  0.5216252 ]
  [0.60390383 0.5450803  0.5333156 ]
  [0.6039216  0.54509807 0.53333336]
  ...
  [0.53443325 0.46518335 0.45689413]
  [0.5062618  0.4356914  0.42784232]
  [0.5119619  0.44137365 0.4335305 ]]
```

```
 [[0.59468484 0.5358613  0.5240966 ]
  [0.6025832  0.5437597  0.531995  ]
  [0.6039216  0.54509807 0.53333336]
  ...
  [0.5541919  0.48360366 0.47576052]
  [0.5603544  0.48976615 0.481923  ]
  [0.56569797 0.4951097  0.48726657]]

 ...

 [[0.48668402 0.41609576 0.40825263]
  [0.48642808 0.41583985 0.4079967 ]
  [0.488629   0.41804075 0.41019762]
  ...
  [0.54698163 0.49207968 0.4881581 ]
  [0.55721545 0.5023135  0.49749973]
  [0.52983314 0.47493115 0.46316645]]

 [[0.50371116 0.4331229  0.42527977]
  [0.5058883  0.43531787 0.42746878]
  [0.50632846 0.4370786  0.42878932]
  ...
  [0.5452209  0.49031895 0.4863974 ]
  [0.5607369  0.50583494 0.5019015 ]
  [0.52956265 0.47466066 0.46293366]]

 [[0.50934494 0.44914427 0.4378386 ]
  [0.5097851  0.450905   0.43915915]
  [0.51317453 0.45435104 0.44258633]
  ...
  [0.5442791  0.48937717 0.4854556 ]
  [0.55904734 0.5041454  0.5002238 ]
  [0.5330841  0.47818208 0.46733546]]]
```

```
validation_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
    horizontal_flip=True)
validation_generator = validation_datagen.flow_from_directory(
    'datasplit/val',
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=32,
    class_mode="sparse"
)
```

```
Found 1597 images belonging to 10 classes.
```

```
test_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=10,
```

```
    horizontal_flip=True)

test_generator = test_datagen.flow_from_directory(
    'datasplit/test',
    target_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=32,
    class_mode="sparse"
)
```

```
Found 1610 images belonging to 10 classes.
```

```
for image_batch, label_batch in test_generator:
    print(image_batch[0])
    break
```

```
[[[0.6917023  0.63680035 0.6328788 ]
  [0.7403225  0.68542045 0.6814989 ]
  [0.73122096 0.676319   0.67239743]
  ...
  [0.7706709  0.71184736 0.6843964 ]
  [0.73305154 0.674228   0.64677703]
  [0.7184002  0.65957665 0.6321257 ]]

 [[0.7162291  0.6613271  0.65740556]
  [0.7449725  0.69007057 0.686149  ]
  [0.7193429  0.6644409  0.6605193 ]
  ...
  [0.7476592  0.6888357  0.6613847 ]
  [0.7076574  0.6488339  0.6213829 ]
  [0.7326814  0.67385787 0.6464069 ]]

 [[0.6787104  0.62380844 0.6198869 ]
  [0.7178243  0.6629223  0.65900075]
  [0.7278865  0.67298454 0.669063  ]
  ...
  [0.81113696 0.75231344 0.72486246]
  [0.7069211  0.6480976  0.6206466 ]
  [0.7017587  0.64293516 0.6154842 ]]

 ...

 [[0.52146757 0.44303623 0.41558522]
  [0.5540304  0.47559908 0.4481481 ]
  [0.5676182  0.48918685 0.46173587]
  ...
  [0.5436959  0.43389204 0.3907548 ]
  [0.54015815 0.43035424 0.387217  ]
  [0.5288785  0.4190746  0.37593734]]

 [[0.5146807  0.4362493  0.4087983 ]
  [0.55777156 0.47934017 0.4518892 ]
```

```
  [0.55799615 0.47956476 0.45211378]
  ...
  [0.49844077 0.38863683 0.34549958]
  [0.5264226  0.4166187  0.37348145]
  [0.56984955 0.4600456  0.41690835]]

 [[0.6234835  0.5450521  0.51760113]
  [0.57302624 0.49459493 0.46714392]
  [0.55540043 0.47696906 0.44951808]
  ...
  [0.56852645 0.4587225  0.41558522]
  [0.55877304 0.4489691  0.40583184]
  [0.5568628  0.44705886 0.4039216 ]]]
```

Building the Model

```
input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 10

model = models.Sequential([
    layers.InputLayer(input_shape=input_shape),
    layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      896       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 127, 127, 32)     0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496     
```

```
 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 64)        36928

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 12, 12, 64)        36928

 max_pooling2d_4 (MaxPooling  (None, 6, 6, 64)         0
 2D)

 conv2d_5 (Conv2D)           (None, 4, 4, 64)          36928

 max_pooling2d_5 (MaxPooling  (None, 2, 2, 64)         0
 2D)

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 64)                16448

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 184,202
Trainable params: 184,202
Non-trainable params: 0
_________________________________________________________________
```

Compiling the Model

We use adam Optimizer, SparseCategoricalCrossentropy for losses, accuracy as a metric

```
model.compile(
    optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)

# 12804 train images divide by batch size (32) equal to steps per
epoch
```

```
12804 / 32
```

```
400.125
```

```
# 1597 validation images divide by batch size (32) equal to validation
steps per epoch

1597 / 32
```

```
49.90625
```

```
acc = []
val_acc = []
loss = []
val_loss = []

for epoch in range(EPOCHS):
    print(f"Epoch {epoch + 1}/{EPOCHS}:")

    history = model.fit(
        train_generator,
        steps_per_epoch=400,
        batch_size=BATCH_SIZE,
        validation_data=validation_generator,
        validation_steps=49,
        verbose=1,
        epochs=1,  # Train for 1 epoch at a time
    )

    # Accumulate the history across epochs
    acc.append(history.history['accuracy'])
    val_acc.append(history.history['val_accuracy'])
    loss.append(history.history['loss'])
    val_loss.append(history.history['val_loss'])
```

```
Epoch 1/50:
400/400 [==============================] - 1494s 4s/step - loss:
1.5617 - accuracy: 0.4468 - val_loss: 1.0229 - val_accuracy: 0.6460
Epoch 2/50:
400/400 [==============================] - 1442s 4s/step - loss:
0.7526 - accuracy: 0.7404 - val_loss: 0.6371 - val_accuracy: 0.7902
Epoch 3/50:
400/400 [==============================] - 1412s 4s/step - loss:
0.5050 - accuracy: 0.8255 - val_loss: 0.5138 - val_accuracy: 0.8253
Epoch 4/50:
400/400 [==============================] - 1371s 3s/step - loss:
0.3946 - accuracy: 0.8620 - val_loss: 0.2941 - val_accuracy: 0.9126
Epoch 5/50:
400/400 [==============================] - 1343s 3s/step - loss:
0.3309 - accuracy: 0.8842 - val_loss: 0.3176 - val_accuracy: 0.8916
```

```
Epoch 6/50:
400/400 [==============================] - 1353s 3s/step - loss:
0.2748 - accuracy: 0.9024 - val_loss: 0.3154 - val_accuracy: 0.9082
Epoch 7/50:
400/400 [==============================] - 1351s 3s/step - loss:
0.2481 - accuracy: 0.9150 - val_loss: 0.3279 - val_accuracy: 0.8890
Epoch 8/50:
400/400 [==============================] - 1342s 3s/step - loss:
0.2216 - accuracy: 0.9256 - val_loss: 0.2295 - val_accuracy: 0.9209
Epoch 9/50:
400/400 [==============================] - 1368s 3s/step - loss:
0.1851 - accuracy: 0.9356 - val_loss: 0.1823 - val_accuracy: 0.9362
Epoch 10/50:
400/400 [==============================] - 1353s 3s/step - loss:
0.1720 - accuracy: 0.9410 - val_loss: 0.1977 - val_accuracy: 0.9305
Epoch 11/50:
400/400 [==============================] - 1336s 3s/step - loss:
0.1429 - accuracy: 0.9488 - val_loss: 0.1706 - val_accuracy: 0.9413
Epoch 12/50:
400/400 [==============================] - 1343s 3s/step - loss:
0.1622 - accuracy: 0.9443 - val_loss: 0.1756 - val_accuracy: 0.9388
Epoch 13/50:
400/400 [==============================] - 1323s 3s/step - loss:
0.1203 - accuracy: 0.9584 - val_loss: 0.1632 - val_accuracy: 0.9420
Epoch 14/50:
400/400 [==============================] - 1336s 3s/step - loss:
0.1243 - accuracy: 0.9577 - val_loss: 0.1912 - val_accuracy: 0.9394
Epoch 15/50:
400/400 [==============================] - 1330s 3s/step - loss:
0.1254 - accuracy: 0.9573 - val_loss: 0.3450 - val_accuracy: 0.8897
Epoch 16/50:
400/400 [==============================] - 1322s 3s/step - loss:
0.1131 - accuracy: 0.9591 - val_loss: 0.1831 - val_accuracy: 0.9471
Epoch 17/50:
400/400 [==============================] - 1320s 3s/step - loss:
0.1096 - accuracy: 0.9646 - val_loss: 0.1742 - val_accuracy: 0.9509
Epoch 18/50:
400/400 [==============================] - 1318s 3s/step - loss:
0.1022 - accuracy: 0.9647 - val_loss: 0.1214 - val_accuracy: 0.9579
Epoch 19/50:
400/400 [==============================] - 1318s 3s/step - loss:
0.1002 - accuracy: 0.9651 - val_loss: 0.1786 - val_accuracy: 0.9420
Epoch 20/50:
400/400 [==============================] - 1333s 3s/step - loss:
0.0794 - accuracy: 0.9724 - val_loss: 0.1783 - val_accuracy: 0.9464
Epoch 21/50:
400/400 [==============================] - 1317s 3s/step - loss:
0.1073 - accuracy: 0.9634 - val_loss: 0.1587 - val_accuracy: 0.9452
Epoch 22/50:
```

```
400/400 [==============================] - 1305s 3s/step - loss:
0.0837 - accuracy: 0.9695 - val_loss: 0.1307 - val_accuracy: 0.9592
Epoch 23/50:
400/400 [==============================] - 1579s 4s/step - loss:
0.0671 - accuracy: 0.9769 - val_loss: 0.1001 - val_accuracy: 0.9688
Epoch 24/50:
400/400 [==============================] - 1708s 4s/step - loss:
0.0800 - accuracy: 0.9720 - val_loss: 0.2718 - val_accuracy: 0.9177
Epoch 25/50:
400/400 [==============================] - 1647s 4s/step - loss:
0.0919 - accuracy: 0.9679 - val_loss: 0.1149 - val_accuracy: 0.9624
Epoch 26/50:
400/400 [==============================] - 1472s 4s/step - loss:
0.0615 - accuracy: 0.9793 - val_loss: 0.1651 - val_accuracy: 0.9605
Epoch 27/50:
400/400 [==============================] - 1724s 4s/step - loss:
0.0771 - accuracy: 0.9737 - val_loss: 0.1517 - val_accuracy: 0.9534
Epoch 28/50:
400/400 [==============================] - 1523s 4s/step - loss:
0.0699 - accuracy: 0.9753 - val_loss: 0.1264 - val_accuracy: 0.9675
Epoch 29/50:
400/400 [==============================] - 1726s 4s/step - loss:
0.0601 - accuracy: 0.9798 - val_loss: 0.1306 - val_accuracy: 0.9598
Epoch 30/50:
400/400 [==============================] - 1505s 4s/step - loss:
0.0602 - accuracy: 0.9803 - val_loss: 0.1182 - val_accuracy: 0.9643
Epoch 31/50:
400/400 [==============================] - 1565s 4s/step - loss:
0.0672 - accuracy: 0.9770 - val_loss: 0.1380 - val_accuracy: 0.9560
Epoch 32/50:
400/400 [==============================] - 1487s 4s/step - loss:
0.0670 - accuracy: 0.9778 - val_loss: 0.1922 - val_accuracy: 0.9541
Epoch 33/50:
400/400 [==============================] - 1493s 4s/step - loss:
0.0333 - accuracy: 0.9884 - val_loss: 0.1543 - val_accuracy: 0.9611
Epoch 34/50:
400/400 [==============================] - 1478s 4s/step - loss:
0.0709 - accuracy: 0.9771 - val_loss: 0.2042 - val_accuracy: 0.9413
Epoch 35/50:
400/400 [==============================] - 1478s 4s/step - loss:
0.0612 - accuracy: 0.9781 - val_loss: 0.1145 - val_accuracy: 0.9649
Epoch 36/50:
400/400 [==============================] - 1506s 4s/step - loss:
0.0438 - accuracy: 0.9841 - val_loss: 0.1239 - val_accuracy: 0.9624
Epoch 37/50:
400/400 [==============================] - 1505s 4s/step - loss:
0.0701 - accuracy: 0.9770 - val_loss: 0.2012 - val_accuracy: 0.9464
Epoch 38/50:
400/400 [==============================] - 1480s 4s/step - loss:
```

```
0.0436 - accuracy: 0.9858 - val_loss: 0.1222 - val_accuracy: 0.9700
Epoch 39/50:
400/400 [==============================] - 1486s 4s/step - loss:
0.0549 - accuracy: 0.9825 - val_loss: 0.1039 - val_accuracy: 0.9656
Epoch 40/50:
400/400 [==============================] - 1487s 4s/step - loss:
0.0513 - accuracy: 0.9832 - val_loss: 0.1350 - val_accuracy: 0.9579
Epoch 41/50:
400/400 [==============================] - 1555s 4s/step - loss:
0.0693 - accuracy: 0.9779 - val_loss: 0.1359 - val_accuracy: 0.9585
Epoch 42/50:
400/400 [==============================] - 2002s 5s/step - loss:
0.0468 - accuracy: 0.9858 - val_loss: 0.1067 - val_accuracy: 0.9668
Epoch 43/50:
400/400 [==============================] - 1985s 5s/step - loss:
0.0344 - accuracy: 0.9890 - val_loss: 0.2404 - val_accuracy: 0.9337
Epoch 44/50:
400/400 [==============================] - 2056s 5s/step - loss:
0.0522 - accuracy: 0.9824 - val_loss: 0.1191 - val_accuracy: 0.9643
Epoch 45/50:
400/400 [==============================] - 1486s 4s/step - loss:
0.0436 - accuracy: 0.9846 - val_loss: 0.1505 - val_accuracy: 0.9617
Epoch 46/50:
400/400 [==============================] - 1468s 4s/step - loss:
0.0616 - accuracy: 0.9796 - val_loss: 0.1469 - val_accuracy: 0.9649
Epoch 47/50:
400/400 [==============================] - 1469s 4s/step - loss:
0.0258 - accuracy: 0.9928 - val_loss: 0.1418 - val_accuracy: 0.9585
Epoch 48/50:
400/400 [==============================] - 1479s 4s/step - loss:
0.0485 - accuracy: 0.9846 - val_loss: 0.1256 - val_accuracy: 0.9668
Epoch 49/50:
400/400 [==============================] - 1492s 4s/step - loss:
0.0544 - accuracy: 0.9811 - val_loss: 0.2308 - val_accuracy: 0.9420
Epoch 50/50:
400/400 [==============================] - 1483s 4s/step - loss:
0.0424 - accuracy: 0.9858 - val_loss: 0.1105 - val_accuracy: 0.9700
```

```
scores = model.evaluate(test_generator)
```

```
51/51 [==============================] - 81s 2s/step - loss: 0.1470 -
accuracy: 0.9671
```

```
# Scores is just a list containing loss and accuracy value

scores
```

```
[0.14696912467479706, 0.9670807719230652]
```

Plotting the Accuracy and Loss Curves

```
history
```

```
<keras.callbacks.History at 0x1bc96a0c430>
```

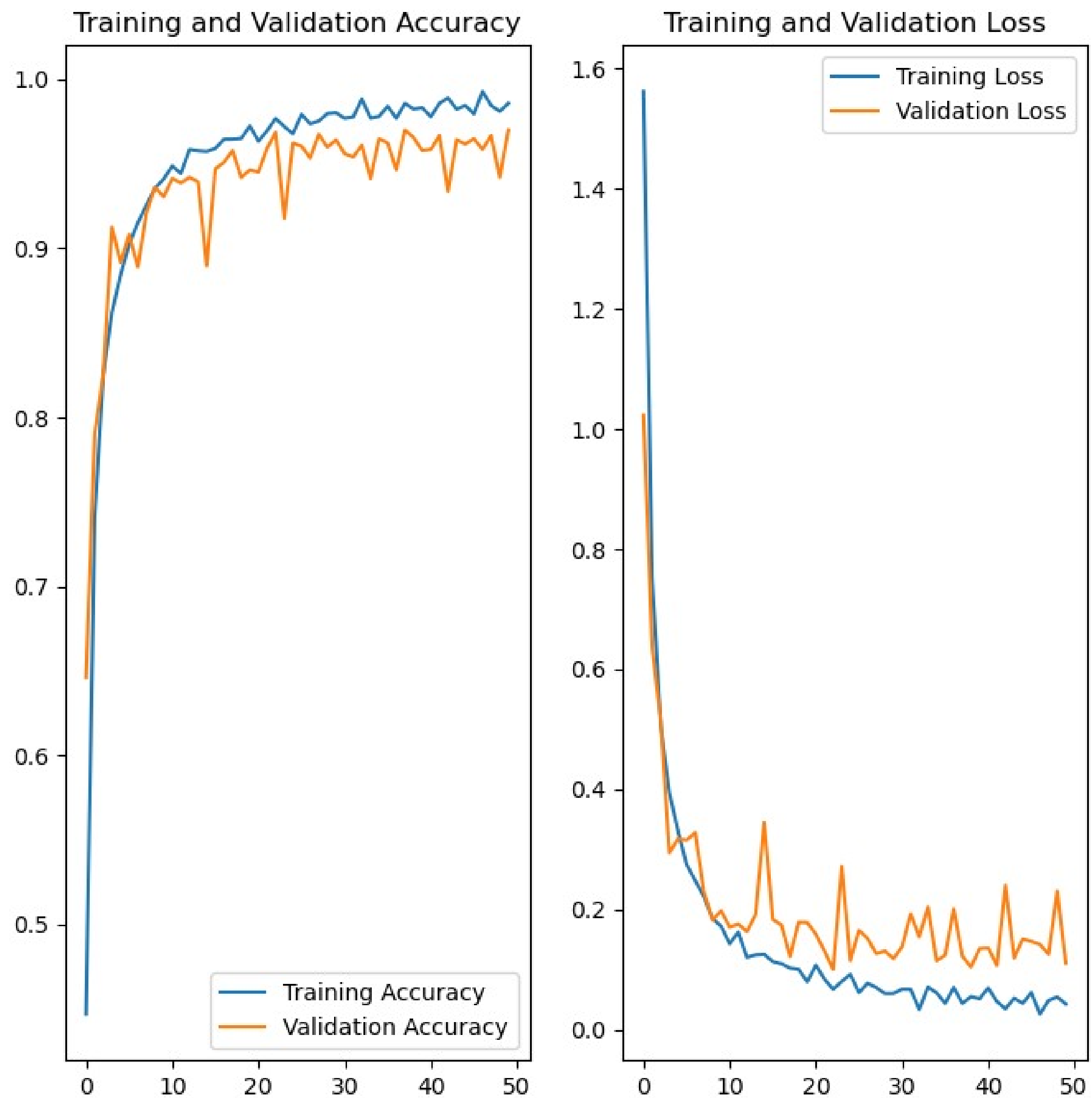
```
history.params
```

```
{'verbose': 1, 'epochs': 1, 'steps': 400}
```

```
history.history.keys()

# loss, accuracy, val loss etc are a python list containing values of
loss, accuracy etc at the end of each epoch
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label='Training Loss')
plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

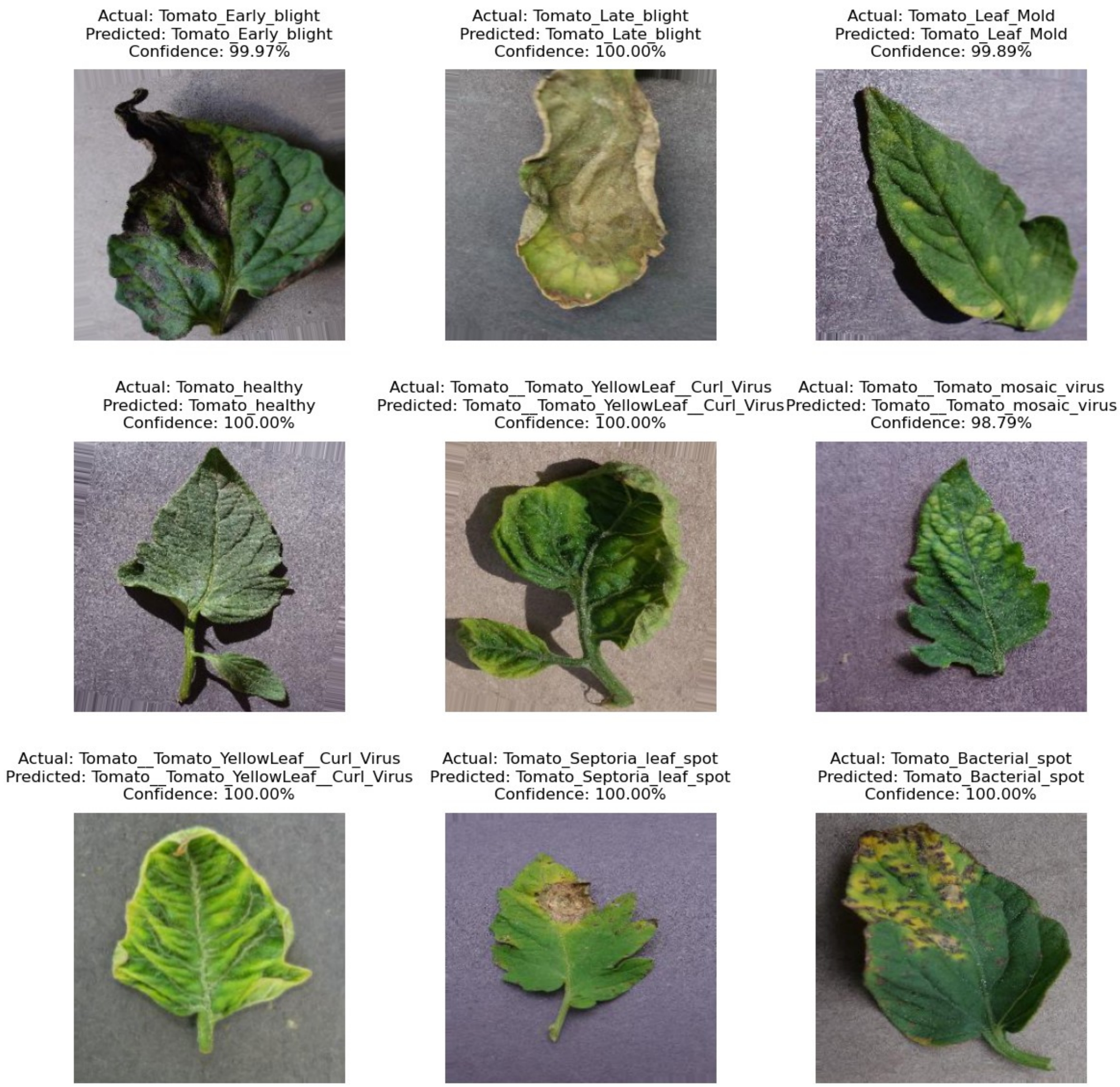


Run prediction on sample images

```
import random

# Assuming test_generator is your data generator
fig, axs = plt.subplots(3, 3, figsize=(12, 12))

# Get the total number of batches
num_batches = len(test_generator)

# Select a random batch index
random_batch_index = random.randint(0, num_batches)
```

```
# Load the random batch
image_batch, label_batch = test_generator[random_batch_index]

# Predictions for the random batch
batch_predictions = model.predict(image_batch)

# Randomly select 9 images from the batch
random_indices = random.sample(range(len(image_batch)), 9)

for i, index in enumerate(random_indices):
    row = i // 3
    col = i % 3

    image = image_batch[index]
    label = int(label_batch[index])
    prediction = np.argmax(batch_predictions[index])
    # Confidence percentage
    confidence = np.max(batch_predictions[index]) * 100

    axs[row, col].imshow(image)
    axs[row, col].set_title(f"Actual: {class_names[label]}\nPredicted: {class_names[prediction]}\nConfidence: {confidence:.2f}%", pad=10)
    axs[row, col].axis('off')

plt.tight_layout(pad=3.0)
plt.show()
```

```
1/1 [==============================] - 1s 616ms/step
```



Saving the Model

```python
import os
from datetime import datetime

# Keras & H5 format
# Create directories if they don't exist
keras_directory = "model/keras"
h5_directory = "model/h5"

os.makedirs(keras_directory, exist_ok=True)
os.makedirs(h5_directory, exist_ok=True)

now = datetime.now()
```

```
date_time_str = now.strftime("%Y-%m-%d_%H-%M-%S")

keras_file_name = os.path.join(keras_directory,
f"tomato_{date_time_str}.keras")
h5_file_name = os.path.join(h5_directory,
f"tomato_{date_time_str}.h5")

model.save(keras_file_name)
print("Model (keras) saved with file name:", keras_file_name)

model.save(h5_file_name)
print("Model (h5) saved with file name:", h5_file_name)

# Tflite format

converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

tflite_folder = "model/tflite"
os.makedirs(tflite_folder, exist_ok=True)
tflite_file_name = os.path.join(tflite_folder,
f"tomato_{date_time_str}.tflite")
with open(tflite_file_name, "wb") as f:
    f.write(tflite_model)

print("Model (TensorFlow Lite) saved with file name:",
tflite_file_name)
```

```
Model (keras) saved with file name: model/keras\tomato_2024-05-03_20-
50-08.keras
Model (h5) saved with file name: model/h5\tomato_2024-05-03_20-50-
08.h5

WARNING:absl:Found untraced functions such as
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op, _jit_compiled_convolution_op,
_jit_compiled_convolution_op while saving (showing 5 of 6). These
functions will not be directly callable after loading.

INFO:tensorflow:Assets written to: C:\Users\naimm\AppData\Local\Temp\
tmp6qe8nulk\assets

INFO:tensorflow:Assets written to: C:\Users\naimm\AppData\Local\Temp\
tmp6qe8nulk\assets

Model (TensorFlow Lite) saved with file name: model/tflite\
tomato_2024-05-03_20-50-08.tflite
```