Potato Disease Classification

Import all the Dependencies

In [1]:
```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML
```

```
WARNING:tensorflow:From C:\Users\91760\anaconda3\lib\site-packages\keras\src\losses.p
y:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.
compat.v1.losses.sparse_softmax_cross_entropy instead.
```

Set all the Constants

In [2]:
```
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=50
```

Import data into tensorflow dataset object

In [3]:
```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "training",
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

```
Found 2152 files belonging to 3 classes.
```

In [4]:
```
class_names = dataset.class_names
class_names
```

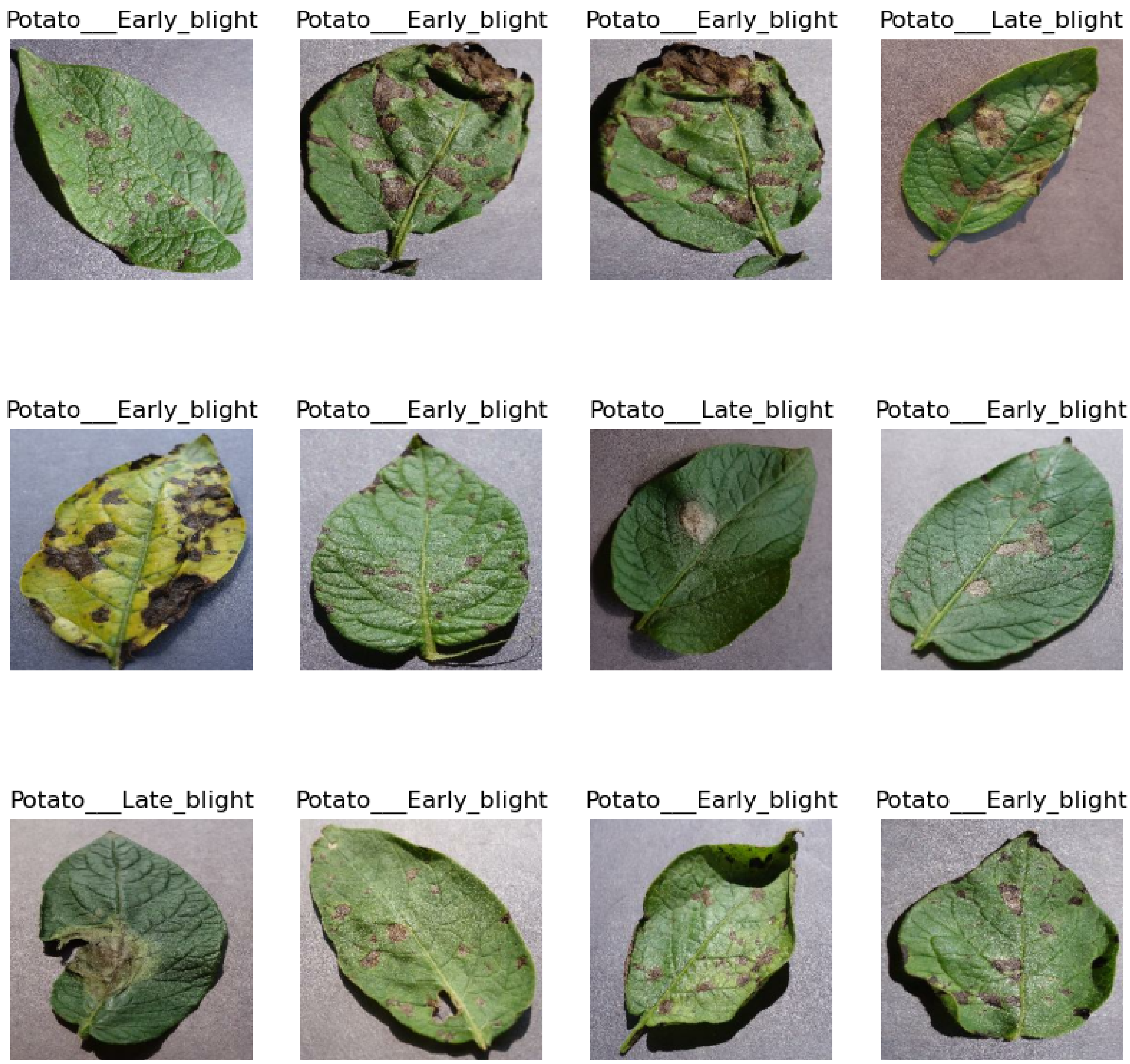
Out[4]:
```
['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
```

In [5]:
```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

```
(32, 256, 256, 3)
[1 1 1 0 0 0 0 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 0 1 0 0 1 1 2 0 0]
```

Visualize some of the images from dataset

In [6]:
```
plt.figure(figsize=(10, 10))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```



Function to Split Dataset

In [7]:
```
len(dataset)
```
Out[7]:
```
68
```

In [8]:
```
train_size = 0.8
len(dataset)*train_size
```
Out[8]:
```
54.400000000000006
```

In [9]:
```
train_ds = dataset.take(54)
len(train_ds)
```
Out[9]:
```
54
```

In [10]:
```
test_ds = dataset.skip(54)
len(test_ds)
```
Out[10]:
```
14
```

In [11]:
```
val_size=0.1
len(dataset)*val_size
```
Out[11]:
```
6.800000000000001
```

In [12]:
```
val_ds = test_ds.take(6)
len(val_ds)
```
Out[12]:
```
6
```
In [13]:
```
test_ds = test_ds.skip(6)
len(test_ds)
```
Out[13]:
```
8
```
In [14]:
```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shu
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)

    return train_ds, val_ds, test_ds
```
In [15]:
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```
In [16]:
```
len(train_ds)
```
Out[16]:
```
54
```
In [17]:
```
len(val_ds)
```
Out[17]:
```
6
```
In [18]:
```
len(test_ds)
```
Out[18]:
```
8
```
Cache, Shuffle, and Prefetch the Dataset

In [19]:
```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```
Building the Model

In [20]:
```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```
```
WARNING:tensorflow:From C:\Users\91760\anaconda3\lib\site-packages\keras\src\backend.
py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_defa
ult_graph instead.
```
Data Augmentation

```
In [21]: data_augmentation = tf.keras.Sequential([
           layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
           layers.experimental.preprocessing.RandomRotation(0.2),
         ])
```

Applying Data Augmentation to Train Dataset

```
In [22]: train_ds = train_ds.map(
             lambda x, y: (data_augmentation(x, training=True), y)
         ).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Model Architecture

```
In [23]: input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
         n_classes = 3

         model = models.Sequential([
             resize_and_rescale,
             layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape
             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.build(input_shape=input_shape)
```

```
WARNING:tensorflow:From C:\Users\91760\anaconda3\lib\site-packages\keras\src\layers\p
ooling\max_pooling2d.py:161: The name tf.nn.max_pool is deprecated. Please use tf.nn.
max_pool2d instead.
```

```
In [24]: model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 conv2d (Conv2D)             (32, 254, 254, 32)        896

 max_pooling2d (MaxPooling2  (32, 127, 127, 32)        0
 D)

 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPoolin  (32, 62, 62, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928

 max_pooling2d_2 (MaxPoolin  (32, 30, 30, 64)          0
 g2D)

 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928
```

```
 max_pooling2d_3 (MaxPoolin  (32, 14, 14, 64)          0
 g2D)

 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928

 max_pooling2d_4 (MaxPoolin  (32, 6, 6, 64)            0
 g2D)

 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928

 max_pooling2d_5 (MaxPoolin  (32, 2, 2, 64)            0
 g2D)

 flatten (Flatten)           (32, 256)                 0

 dense (Dense)               (32, 64)                  16448

 dense_1 (Dense)             (32, 3)                   195

=================================================================
Total params: 183747 (717.76 KB)
Trainable params: 183747 (717.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compiling the Model

In [25]:

```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

```
WARNING:tensorflow:From C:\Users\91760\anaconda3\lib\site-packages\keras\src\optimize
rs\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v
1.train.Optimizer instead.
```

In [26]:

```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=50,
)
```

```
Epoch 1/50
WARNING:tensorflow:From C:\Users\91760\anaconda3\lib\site-packages\keras\src\utils\tf
_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.comp
at.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\91760\anaconda3\lib\site-packages\keras\src\engine\b
ase_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecate
d. Please use tf.compat.v1.executing_eagerly_outside_functions instead.

54/54 [==============================] - 251s 4s/step - loss: 0.8926 - accuracy: 0.51
62 - val_loss: 0.7564 - val_accuracy: 0.6406
Epoch 2/50
54/54 [==============================] - 220s 4s/step - loss: 0.5741 - accuracy: 0.72
63 - val_loss: 0.4077 - val_accuracy: 0.8125
Epoch 3/50
54/54 [==============================] - 1391s 26s/step - loss: 0.3620 - accuracy: 0.
8513 - val_loss: 0.3881 - val_accuracy: 0.8229
Epoch 4/50
54/54 [==============================] - 216s 4s/step - loss: 0.2209 - accuracy: 0.90
45 - val_loss: 0.2484 - val_accuracy: 0.8854
Epoch 5/50
```

```
54/54 [==============================] - 213s 4s/step - loss: 0.1090 - accuracy: 0.94
91 - val_loss: 0.1930 - val_accuracy: 0.9323
Epoch 6/50
54/54 [==============================] - 1020s 19s/step - loss: 0.1622 - accuracy: 0.
9392 - val_loss: 0.1460 - val_accuracy: 0.9479
Epoch 7/50
54/54 [==============================] - 213s 4s/step - loss: 0.1117 - accuracy: 0.96
06 - val_loss: 0.0626 - val_accuracy: 0.9740
Epoch 8/50
54/54 [==============================] - 210s 4s/step - loss: 0.0660 - accuracy: 0.97
69 - val_loss: 0.2849 - val_accuracy: 0.8958
Epoch 9/50
54/54 [==============================] - 212s 4s/step - loss: 0.0751 - accuracy: 0.97
40 - val_loss: 0.1044 - val_accuracy: 0.9479
Epoch 10/50
54/54 [==============================] - 211s 4s/step - loss: 0.0815 - accuracy: 0.97
11 - val_loss: 0.1703 - val_accuracy: 0.9427
Epoch 11/50
54/54 [==============================] - 212s 4s/step - loss: 0.0646 - accuracy: 0.97
57 - val_loss: 0.1237 - val_accuracy: 0.9427
Epoch 12/50
54/54 [==============================] - 221s 4s/step - loss: 0.0960 - accuracy: 0.96
24 - val_loss: 0.0269 - val_accuracy: 0.9948
Epoch 13/50
54/54 [==============================] - 214s 4s/step - loss: 0.0543 - accuracy: 0.97
86 - val_loss: 0.0261 - val_accuracy: 0.9896
Epoch 14/50
54/54 [==============================] - 212s 4s/step - loss: 0.0508 - accuracy: 0.97
80 - val_loss: 0.2158 - val_accuracy: 0.9323
Epoch 15/50
54/54 [==============================] - 211s 4s/step - loss: 0.0368 - accuracy: 0.98
73 - val_loss: 0.0639 - val_accuracy: 0.9635
Epoch 16/50
54/54 [==============================] - 171s 3s/step - loss: 0.0516 - accuracy: 0.98
03 - val_loss: 0.8642 - val_accuracy: 0.8281
Epoch 17/50
54/54 [==============================] - 176s 3s/step - loss: 0.0932 - accuracy: 0.96
64 - val_loss: 0.0656 - val_accuracy: 0.9635
Epoch 18/50
54/54 [==============================] - 178s 3s/step - loss: 0.0403 - accuracy: 0.98
61 - val_loss: 0.0632 - val_accuracy: 0.9792
Epoch 19/50
54/54 [==============================] - 190s 4s/step - loss: 0.0280 - accuracy: 0.99
02 - val_loss: 0.0261 - val_accuracy: 0.9792
Epoch 20/50
54/54 [==============================] - 184s 3s/step - loss: 0.0372 - accuracy: 0.98
90 - val_loss: 0.0876 - val_accuracy: 0.9635
Epoch 21/50
54/54 [==============================] - 173s 3s/step - loss: 0.0514 - accuracy: 0.98
03 - val_loss: 0.1123 - val_accuracy: 0.9635
Epoch 22/50
54/54 [==============================] - 177s 3s/step - loss: 0.0450 - accuracy: 0.98
15 - val_loss: 0.0742 - val_accuracy: 0.9531
Epoch 23/50
54/54 [==============================] - 177s 3s/step - loss: 0.0896 - accuracy: 0.96
93 - val_loss: 0.0531 - val_accuracy: 0.9688
Epoch 24/50
54/54 [==============================] - 159s 3s/step - loss: 0.0237 - accuracy: 0.99
25 - val_loss: 0.0054 - val_accuracy: 1.0000
Epoch 25/50
54/54 [==============================] - 148s 3s/step - loss: 0.0238 - accuracy: 0.99
31 - val_loss: 0.0289 - val_accuracy: 0.9844
Epoch 26/50
54/54 [==============================] - 152s 3s/step - loss: 0.0530 - accuracy: 0.98
50 - val_loss: 0.5365 - val_accuracy: 0.8646
Epoch 27/50
54/54 [==============================] - 154s 3s/step - loss: 0.0415 - accuracy: 0.98
55 - val_loss: 0.0485 - val_accuracy: 0.9740
Epoch 28/50
```

```
54/54 [==============================] - 148s 3s/step - loss: 0.0325 - accuracy: 0.98
78 - val_loss: 0.1546 - val_accuracy: 0.9479
Epoch 29/50
54/54 [==============================] - 193s 4s/step - loss: 0.0397 - accuracy: 0.98
55 - val_loss: 0.1268 - val_accuracy: 0.9635
Epoch 30/50
54/54 [==============================] - 198s 4s/step - loss: 0.0288 - accuracy: 0.98
96 - val_loss: 0.0237 - val_accuracy: 0.9948
Epoch 31/50
54/54 [==============================] - 165s 3s/step - loss: 0.0205 - accuracy: 0.99
36 - val_loss: 0.0031 - val_accuracy: 1.0000
Epoch 32/50
54/54 [==============================] - 193s 4s/step - loss: 0.0363 - accuracy: 0.98
44 - val_loss: 0.1079 - val_accuracy: 0.9583
Epoch 33/50
54/54 [==============================] - 157s 3s/step - loss: 0.0338 - accuracy: 0.98
84 - val_loss: 0.1397 - val_accuracy: 0.9583
Epoch 34/50
54/54 [==============================] - 155s 3s/step - loss: 0.0396 - accuracy: 0.98
67 - val_loss: 0.0273 - val_accuracy: 0.9844
Epoch 35/50
54/54 [==============================] - 150s 3s/step - loss: 0.0358 - accuracy: 0.98
96 - val_loss: 0.0297 - val_accuracy: 0.9844
Epoch 36/50
54/54 [==============================] - 151s 3s/step - loss: 0.0533 - accuracy: 0.98
09 - val_loss: 0.0136 - val_accuracy: 0.9948
Epoch 37/50
54/54 [==============================] - 150s 3s/step - loss: 0.0377 - accuracy: 0.98
73 - val_loss: 0.0376 - val_accuracy: 0.9792
Epoch 38/50
54/54 [==============================] - 150s 3s/step - loss: 0.0176 - accuracy: 0.99
42 - val_loss: 0.0511 - val_accuracy: 0.9740
Epoch 39/50
54/54 [==============================] - 149s 3s/step - loss: 0.0152 - accuracy: 0.99
48 - val_loss: 0.0268 - val_accuracy: 0.9896
Epoch 40/50
54/54 [==============================] - 147s 3s/step - loss: 0.0264 - accuracy: 0.99
07 - val_loss: 0.0479 - val_accuracy: 0.9740
Epoch 41/50
54/54 [==============================] - 147s 3s/step - loss: 0.0145 - accuracy: 0.99
77 - val_loss: 0.0143 - val_accuracy: 0.9948
Epoch 42/50
54/54 [==============================] - 147s 3s/step - loss: 0.0158 - accuracy: 0.99
48 - val_loss: 0.0991 - val_accuracy: 0.9583
Epoch 43/50
54/54 [==============================] - 151s 3s/step - loss: 0.0567 - accuracy: 0.98
32 - val_loss: 0.0796 - val_accuracy: 0.9635
Epoch 44/50
54/54 [==============================] - 149s 3s/step - loss: 0.0299 - accuracy: 0.98
84 - val_loss: 0.0452 - val_accuracy: 0.9688
Epoch 45/50
54/54 [==============================] - 151s 3s/step - loss: 0.0247 - accuracy: 0.99
07 - val_loss: 0.0528 - val_accuracy: 0.9740
Epoch 46/50
54/54 [==============================] - 166s 3s/step - loss: 0.0421 - accuracy: 0.98
61 - val_loss: 0.0463 - val_accuracy: 0.9896
Epoch 47/50
54/54 [==============================] - 154s 3s/step - loss: 0.0231 - accuracy: 0.99
25 - val_loss: 0.0148 - val_accuracy: 0.9948
Epoch 48/50
54/54 [==============================] - 160s 3s/step - loss: 0.0463 - accuracy: 0.98
38 - val_loss: 0.2533 - val_accuracy: 0.9219
Epoch 49/50
54/54 [==============================] - 152s 3s/step - loss: 0.0301 - accuracy: 0.98
78 - val_loss: 0.2292 - val_accuracy: 0.9219
Epoch 50/50
54/54 [==============================] - 151s 3s/step - loss: 0.0183 - accuracy: 0.99
42 - val_loss: 0.0176 - val_accuracy: 0.9948
```

In [27]:
```
scores = model.evaluate(test_ds)
```

```
8/8 [==============================] - 10s 622ms/step - loss: 0.0389 - accuracy: 0.98
44
```

In [28]:
```
scores
```

Out[28]:
```
[0.03888298571109772, 0.984375]
```

In [29]:
```
history
```

Out[29]:
```
<keras.src.callbacks.History at 0x1c92f96f910>
```

In [30]:
```
history.params
```

Out[30]:
```
{'verbose': 1, 'epochs': 50, 'steps': 54}
```

In [31]:
```
history.history.keys()
```

Out[31]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [32]:
```
type(history.history['loss'])
```

Out[32]:
```
list
```

In [33]:
```
len(history.history['loss'])
```

Out[33]:
```
50
```

In [34]:
```
history.history['loss'][:5] # show loss for first 5 epochs
```

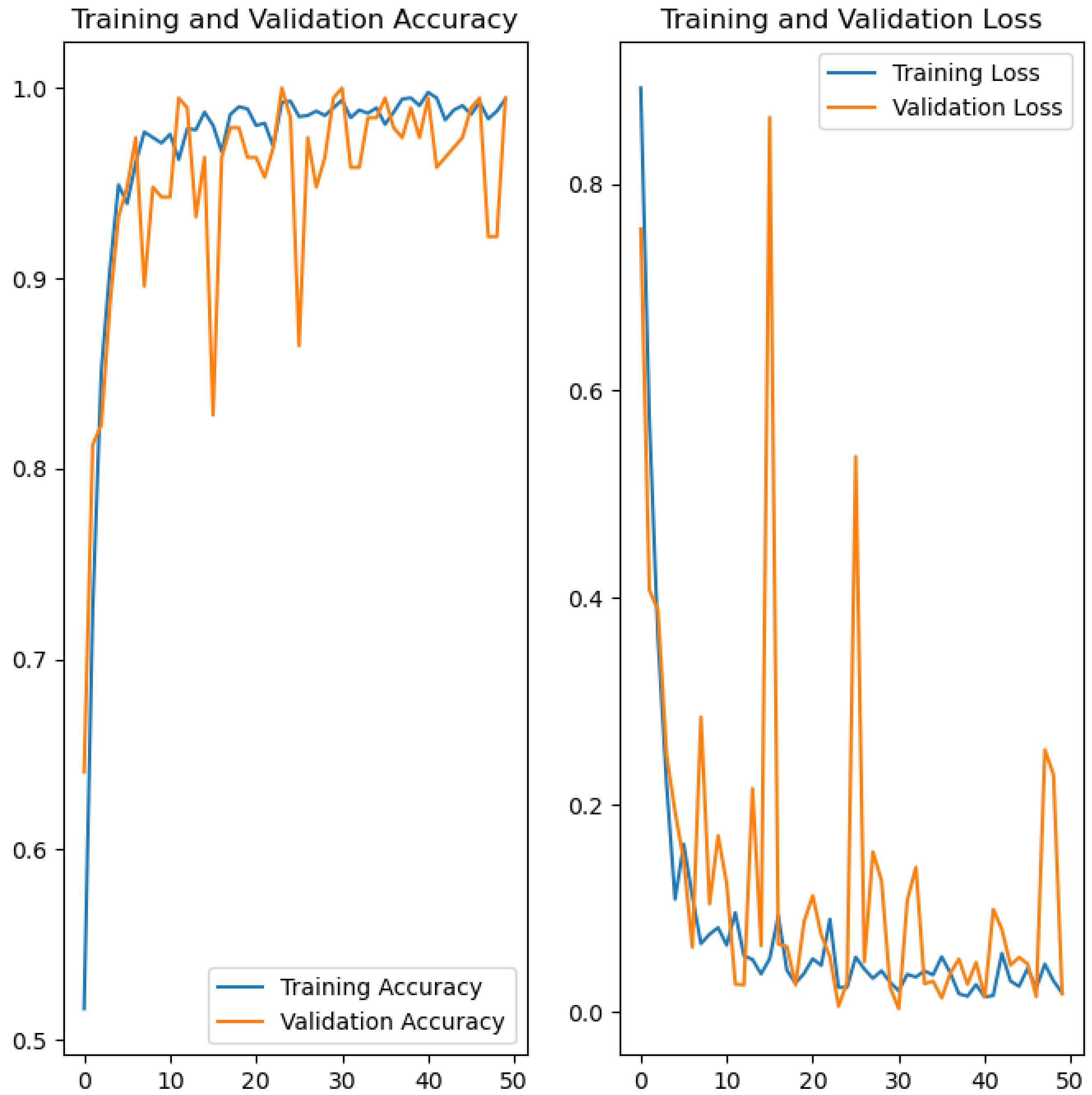
Out[34]:
```
[0.8925877809524536,
 0.5740604400634766,
 0.3619817793369293,
 0.22088845074176788,
 0.10896806418895721]
```

In [35]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']
```

In [36]:
```
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 a sample image

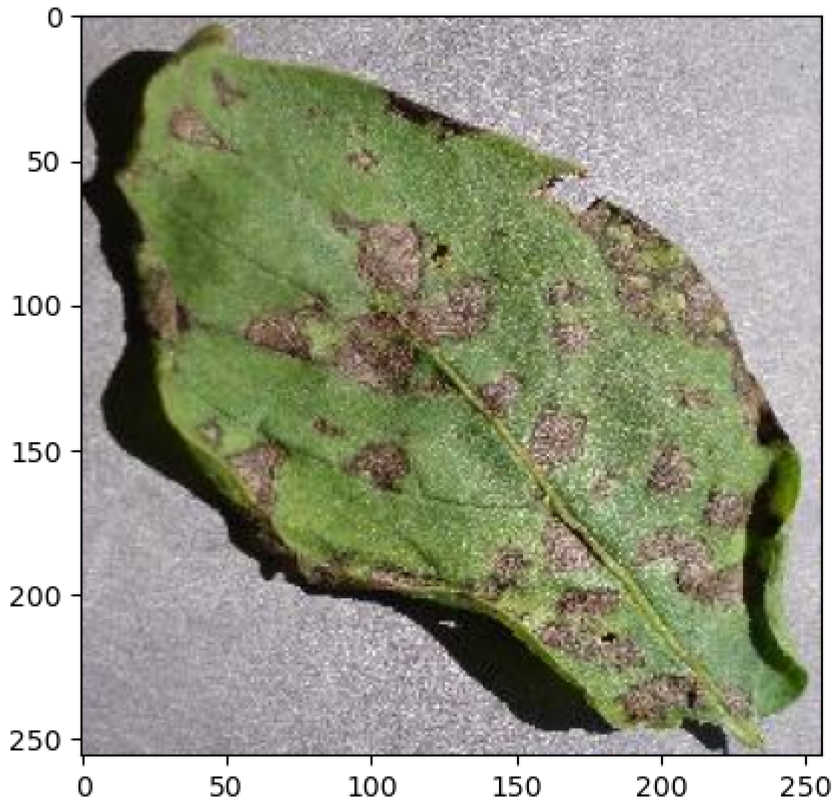
In [37]:
```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])

    batch_prediction = model.predict(images_batch)
    print("predicted label:",class_names[np.argmax(batch_prediction[0])])
```

```
first image to predict
actual label: Potato___Early_blight
1/1 [==============================] - 2s 2s/step
predicted label: Potato___Early_blight
```



Write a function for inference

In [38]:
```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence
```

Now run inference on few sample images

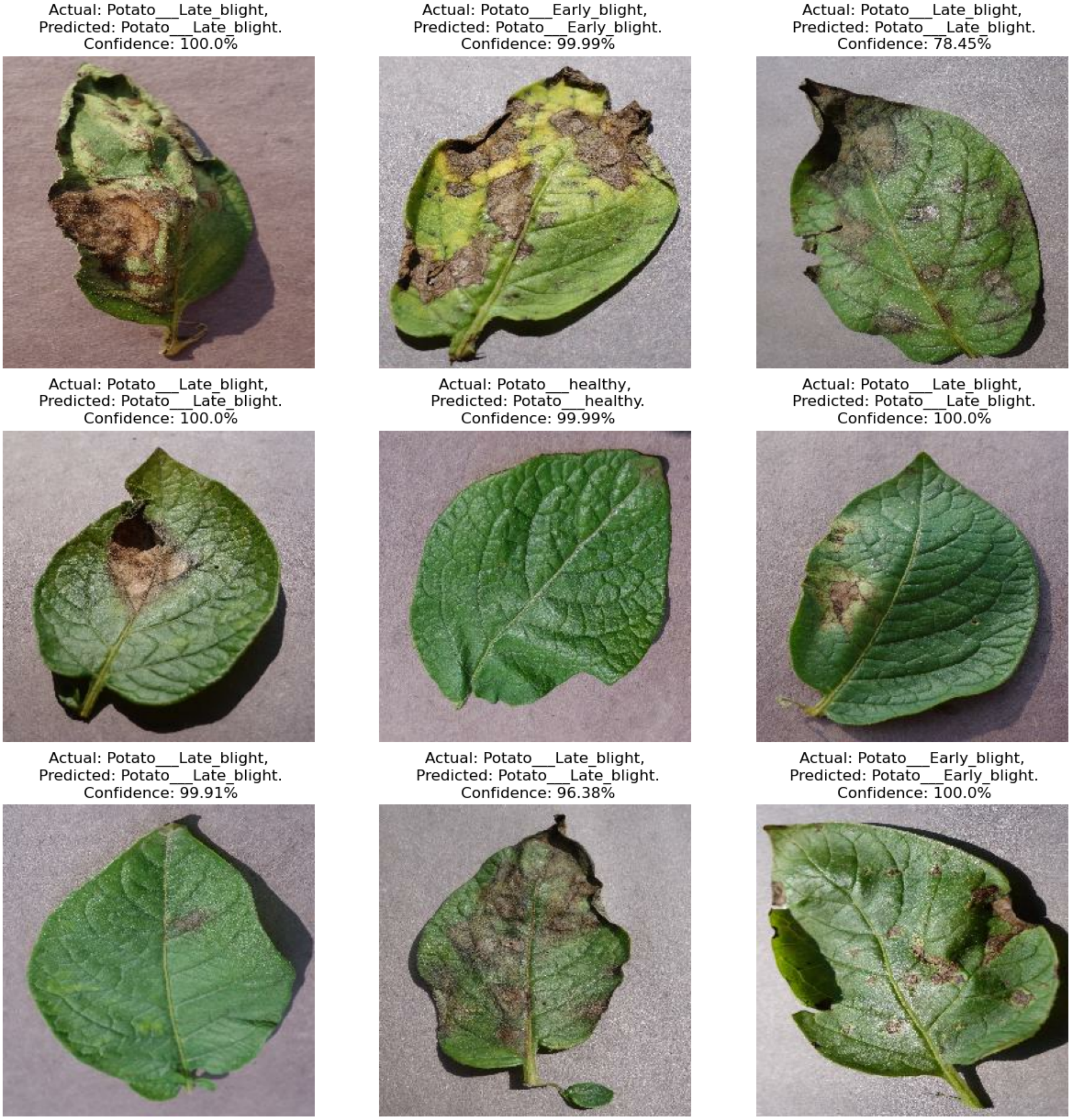
In [39]:
```
plt.figure(figsize=(15, 15))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confiden

        plt.axis("off")
```

```
1/1 [==============================] - 0s 301ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 78ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 50ms/step
```


Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 100.0%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 99.99%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 78.45%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 100.0%
Actual: Potato___healthy,
Predicted: Potato___healthy.
Confidence: 99.99%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 100.0%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.91%
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 96.38%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%