In [1]:

```
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
```

In [2]:

```
batch_size = 32
img_height=256
img_width=256
channels=3
epochs=50
```

In [3]:

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
  "trainig",
  shuffle = True,
  image_size=(img_height, img_width),
  batch_size=batch_size
  )
```

```
Found 300 files belonging to 3 classes.
```

In [4]:

```
class_names = dataset.class_names
class_names
```

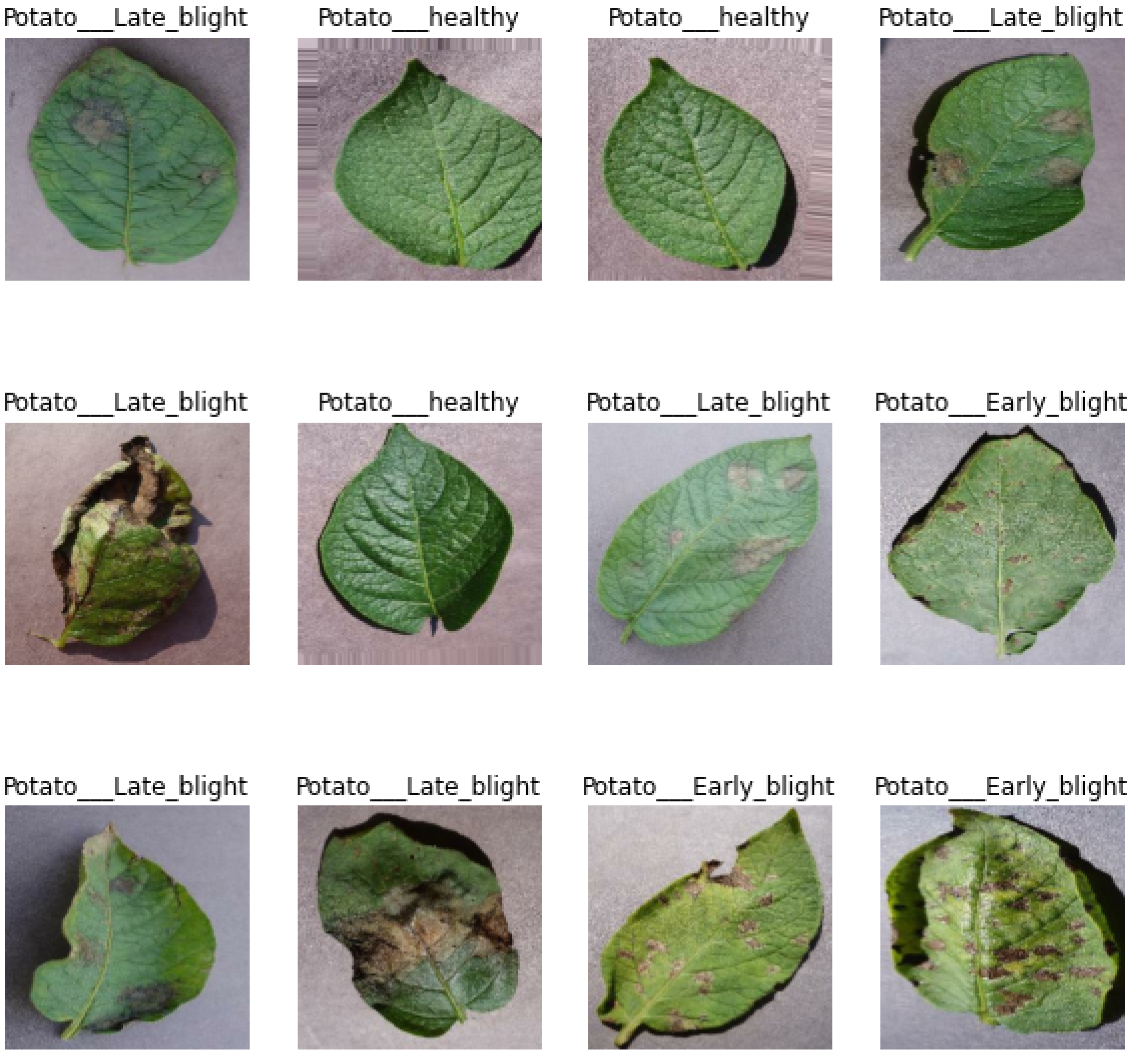
Out[4]:

```
['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
```

In [5]:

```
plt.figure(figsize=(10,10))
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
    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")
```

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



In [ ]:

```

```

In [6]:

```
train_size = 0.6
len(dataset)*train_size
```

Out[6]:

```
6.0
```

In [7]:

```
#train_ds = dataset.take(8)
#len(train_ds)
```

In [8]:

```
#test_ds = dataset.skip(8)
#len(test_ds)
```

In [9]:

```
#val_size=0.1
#len(dataset)*val_size
```

In [10]:

```
#val_ds = test_ds.take(1)
#len(val_ds)
```

In [11]:

```
#test_ds = test_ds.skip(8)
#len(test_ds)
```

In [12]:

```
#test_ds = test_ds.skip(1)
#len(test_ds)
```

In [13]:

```
def get_dataset_partition_tf(ds, train_split=0.8, val_split=0.1,test_split=0.1, shuffle=Tru
    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).take(val_size)
    val_ds = ds.skip(train_size)
    test_ds = ds.skip(train_size).skip(val_size)
    return train_ds, val_ds, test_ds
```

In [14]:

```
train_ds, val_ds, test_ds = get_dataset_partition_tf(dataset)
```

In [15]:

```
len(train_ds)
```

Out[15]:

```
1
```

In [16]:

```
len(val_ds)
```

Out[16]:

```
2
```

In [17]:

```
len(test_ds)
```

Out[17]:

```
1
```

In [18]:

```
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)
```

In [19]:

```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch[0].numpy()/255)
```

```
[[[0.49019608 0.45490196 0.46666667]
  [0.5137255  0.47843137 0.49019608]
  [0.5176471  0.48235294 0.49411765]
  ...
  [0.5803922  0.5568628  0.57254905]
  [0.5882353  0.5647059  0.5803922 ]
  [0.58431375 0.56078434 0.5764706 ]]

 [[0.57254905 0.5372549  0.54901963]
  [0.5803922  0.54509807 0.5568628 ]
  [0.5686275  0.53333336 0.54509807]
  ...
  [0.57254905 0.54901963 0.5647059 ]
  [0.5803922  0.5568628  0.57254905]
  [0.5686275  0.54509807 0.56078434]]

 [[0.5568628  0.52156866 0.53333336]
  [0.5647059  0.5294118  0.5411765 ]
  [0.5647059  0.5294118  0.5411765 ]
  ...
  [0.5686275  0.54509807 0.56078434]
  [0.5686275  0.54509807 0.56078434]
  [0.5568628  0.53333336 0.54901963]]

 ...

 [[0.69411767 0.6745098  0.69803923]
  [0.6862745  0.6666667  0.6901961 ]
  [0.74509805 0.7254902  0.7490196 ]
  ...
  [0.6862745  0.6745098  0.7019608 ]
  [0.7137255  0.7019608  0.7294118 ]
  [0.7372549  0.7254902  0.7529412 ]]

 [[0.65882355 0.6392157  0.6627451 ]
  [0.6117647  0.5921569  0.6156863 ]
  [0.6039216  0.58431375 0.60784316]
  ...
  [0.70980394 0.69803923 0.7254902 ]
  [0.7294118  0.7176471  0.74509805]
  [0.7490196  0.7372549  0.7647059 ]]

 [[0.68235296 0.6627451  0.6862745 ]
  [0.69411767 0.6745098  0.69803923]
  [0.654902   0.63529414 0.65882355]
  ...
  [0.7294118  0.7176471  0.74509805]
  [0.7176471  0.7058824  0.73333335]
  [0.7137255  0.7019608  0.7294118 ]]]
```

In [20]:

```
resize_and_rescale = tf.keras.Sequential([
  layers.Resizing(img_height, img_width),
  layers.Rescaling(1./255)
])
```

In [21]:

```
data_augmentation = tf.keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"),
  layers.RandomRotation(0.2),
])
```

In [22]:

```
input_shape =(batch_size,img_height,img_width,channels)
n_classes = 3

model = models.Sequential([
    resize_and_rescale,
    data_augmentation,
    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,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.Flatten(),
    layers.Dense(64,activation='relu'),
    layers.Dense(n_classes, activation='softmax'),

])
model.build(input_shape=input_shape)
```

In [23]:

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0         
                                                                 
 sequential_1 (Sequential)   (32, 256, 256, 3)         0         
                                                                 
 conv2d (Conv2D)             (32, 254, 254, 32)        896       
                                                                 
 max_pooling2d (MaxPooling2D  (32, 127, 127, 32)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496     
                                                                 
 max_pooling2d_1 (MaxPooling  (32, 62, 62, 64)         0         
 2D)                                                             
                                                                 
 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928     
                                                                 
 max_pooling2d_2 (MaxPooling  (32, 30, 30, 64)         0         
 2D)                                                             
                                                                 
 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928     
                                                                 
 max_pooling2d_3 (MaxPooling  (32, 14, 14, 64)         0         
 2D)                                                             
                                                                 
 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928     
                                                                 
 max_pooling2d_4 (MaxPooling  (32, 6, 6, 64)           0         
 2D)                                                             
                                                                 
 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928     
                                                                 
 max_pooling2d_5 (MaxPooling  (32, 2, 2, 64)           0         
 2D)                                                             
                                                                 
 flatten (Flatten)           (32, 256)                 0         
                                                                 
 dense (Dense)               (32, 64)                  16448     
                                                                 
 dense_1 (Dense)             (32, 3)                   195       
                                                                 
=================================================================
Total params: 183,747
Trainable params: 183,747
Non-trainable params: 0
_________________________________________________________________
```

In [24]:

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

In [25]:

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

```
Epoch 1/50
1/1 [==============================] - 54s 54s/step - loss: 1.1012 - accur
acy: 0.2500 - val_loss: 1.0962 - val_accuracy: 0.3438
Epoch 2/50
1/1 [==============================] - 33s 33s/step - loss: 1.0857 - accur
acy: 0.5000 - val_loss: 1.0970 - val_accuracy: 0.3438
Epoch 3/50
1/1 [==============================] - 21s 21s/step - loss: 1.0526 - accur
acy: 0.5000 - val_loss: 1.1260 - val_accuracy: 0.3438
Epoch 4/50
1/1 [==============================] - 21s 21s/step - loss: 1.0144 - accur
acy: 0.5000 - val_loss: 1.2965 - val_accuracy: 0.3438
Epoch 5/50
1/1 [==============================] - 21s 21s/step - loss: 1.0432 - accur
acy: 0.5000 - val_loss: 1.2150 - val_accuracy: 0.3438
Epoch 6/50
1/1 [==============================] - 23s 23s/step - loss: 1.0080 - accur
acy: 0.5000 - val_loss: 1.1458 - val_accuracy: 0.3438
Epoch 7/50
1/1 [==============================] - 21s 21s/step - loss: 1.0043 - accur
acy: 0.5000 - val_loss: 1.1241 - val_accuracy: 0.3438
Epoch 8/50
1/1 [==============================] - 21s 21s/step - loss: 1.0156 - accur
acy: 0.5000 - val_loss: 1.1204 - val_accuracy: 0.3438
Epoch 9/50
1/1 [==============================] - 22s 22s/step - loss: 1.0175 - accur
acy: 0.5000 - val_loss: 1.1264 - val_accuracy: 0.3438
Epoch 10/50
1/1 [==============================] - 21s 21s/step - loss: 1.0088 - accur
acy: 0.5000 - val_loss: 1.1451 - val_accuracy: 0.3438
Epoch 11/50
1/1 [==============================] - 20s 20s/step - loss: 0.9971 - accur
acy: 0.5000 - val_loss: 1.1823 - val_accuracy: 0.3438
Epoch 12/50
1/1 [==============================] - 15s 15s/step - loss: 0.9941 - accur
acy: 0.5000 - val_loss: 1.2229 - val_accuracy: 0.3438
Epoch 13/50
1/1 [==============================] - 16s 16s/step - loss: 0.9927 - accur
acy: 0.5000 - val_loss: 1.2183 - val_accuracy: 0.3438
Epoch 14/50
1/1 [==============================] - 28s 28s/step - loss: 0.9935 - accur
acy: 0.5000 - val_loss: 1.1741 - val_accuracy: 0.3438
Epoch 15/50
1/1 [==============================] - 23s 23s/step - loss: 0.9808 - accur
acy: 0.5000 - val_loss: 1.1396 - val_accuracy: 0.3438
Epoch 16/50
1/1 [==============================] - 17s 17s/step - loss: 0.9771 - accur
acy: 0.5000 - val_loss: 1.1264 - val_accuracy: 0.3438
Epoch 17/50
1/1 [==============================] - 15s 15s/step - loss: 0.9739 - accur
acy: 0.5000 - val_loss: 1.1308 - val_accuracy: 0.3438
```

```
Epoch 18/50
1/1 [==============================] - 19s 19s/step - loss: 0.9640 - accur
acy: 0.5000 - val_loss: 1.1540 - val_accuracy: 0.3438
Epoch 19/50
1/1 [==============================] - 21s 21s/step - loss: 0.9479 - accur
acy: 0.5000 - val_loss: 1.1831 - val_accuracy: 0.3438
Epoch 20/50
1/1 [==============================] - 17s 17s/step - loss: 0.9404 - accur
acy: 0.5000 - val_loss: 1.1752 - val_accuracy: 0.3438
Epoch 21/50
1/1 [==============================] - 16s 16s/step - loss: 0.9231 - accur
acy: 0.5000 - val_loss: 1.1327 - val_accuracy: 0.3438
Epoch 22/50
1/1 [==============================] - 17s 17s/step - loss: 0.8986 - accur
acy: 0.5000 - val_loss: 1.1261 - val_accuracy: 0.3438
Epoch 23/50
1/1 [==============================] - 16s 16s/step - loss: 0.8687 - accur
acy: 0.5000 - val_loss: 1.1826 - val_accuracy: 0.3438
Epoch 24/50
1/1 [==============================] - 16s 16s/step - loss: 0.8575 - accur
acy: 0.5000 - val_loss: 1.0907 - val_accuracy: 0.3750
Epoch 25/50
1/1 [==============================] - 16s 16s/step - loss: 0.7994 - accur
acy: 0.6562 - val_loss: 1.0642 - val_accuracy: 0.5000
Epoch 26/50
1/1 [==============================] - 16s 16s/step - loss: 0.7726 - accur
acy: 0.7188 - val_loss: 1.1113 - val_accuracy: 0.4688
Epoch 27/50
1/1 [==============================] - 16s 16s/step - loss: 0.7005 - accur
acy: 0.7188 - val_loss: 1.0070 - val_accuracy: 0.6250
Epoch 28/50
1/1 [==============================] - 16s 16s/step - loss: 0.6422 - accur
acy: 0.7812 - val_loss: 1.1442 - val_accuracy: 0.5625
Epoch 29/50
1/1 [==============================] - 16s 16s/step - loss: 0.6003 - accur
acy: 0.7188 - val_loss: 0.8740 - val_accuracy: 0.6250
Epoch 30/50
1/1 [==============================] - 16s 16s/step - loss: 0.5964 - accur
acy: 0.7812 - val_loss: 0.9488 - val_accuracy: 0.6250
Epoch 31/50
1/1 [==============================] - 16s 16s/step - loss: 0.4814 - accur
acy: 0.7812 - val_loss: 1.1619 - val_accuracy: 0.5156
Epoch 32/50
1/1 [==============================] - 16s 16s/step - loss: 0.5324 - accur
acy: 0.7812 - val_loss: 0.8360 - val_accuracy: 0.6406
Epoch 33/50
1/1 [==============================] - 16s 16s/step - loss: 0.4754 - accur
acy: 0.7812 - val_loss: 0.7239 - val_accuracy: 0.7031
Epoch 34/50
1/1 [==============================] - 16s 16s/step - loss: 0.4927 - accur
acy: 0.8125 - val_loss: 1.0641 - val_accuracy: 0.5000
Epoch 35/50
1/1 [==============================] - 16s 16s/step - loss: 0.4098 - accur
acy: 0.8125 - val_loss: 1.1508 - val_accuracy: 0.5625
Epoch 36/50
1/1 [==============================] - 16s 16s/step - loss: 0.4266 - accur
acy: 0.8750 - val_loss: 0.8402 - val_accuracy: 0.6719
Epoch 37/50
1/1 [==============================] - 17s 17s/step - loss: 0.3672 - accur
acy: 0.8438 - val_loss: 0.8993 - val_accuracy: 0.6719
Epoch 38/50
```

```
1/1 [==============================] - 20s 20s/step - loss: 0.3551 - accur
acy: 0.8750 - val_loss: 1.2519 - val_accuracy: 0.4844
Epoch 39/50
1/1 [==============================] - 16s 16s/step - loss: 0.3566 - accur
acy: 0.8438 - val_loss: 0.8449 - val_accuracy: 0.6719
Epoch 40/50
1/1 [==============================] - 16s 16s/step - loss: 0.3347 - accur
acy: 0.8750 - val_loss: 0.7714 - val_accuracy: 0.6875
Epoch 41/50
1/1 [==============================] - 16s 16s/step - loss: 0.3690 - accur
acy: 0.8438 - val_loss: 0.8738 - val_accuracy: 0.6875
Epoch 42/50
1/1 [==============================] - 16s 16s/step - loss: 0.3450 - accur
acy: 0.8750 - val_loss: 1.3364 - val_accuracy: 0.5000
Epoch 43/50
1/1 [==============================] - 16s 16s/step - loss: 0.4310 - accur
acy: 0.7812 - val_loss: 0.7540 - val_accuracy: 0.7031
Epoch 44/50
1/1 [==============================] - 16s 16s/step - loss: 0.2683 - accur
acy: 0.9062 - val_loss: 0.7354 - val_accuracy: 0.6406
Epoch 45/50
1/1 [==============================] - 16s 16s/step - loss: 0.3175 - accur
acy: 0.8438 - val_loss: 0.7142 - val_accuracy: 0.6875
Epoch 46/50
1/1 [==============================] - 16s 16s/step - loss: 0.2483 - accur
acy: 0.9062 - val_loss: 0.9683 - val_accuracy: 0.6094
Epoch 47/50
1/1 [==============================] - 16s 16s/step - loss: 0.3035 - accur
acy: 0.8750 - val_loss: 1.2424 - val_accuracy: 0.5312
Epoch 48/50
1/1 [==============================] - 17s 17s/step - loss: 0.4262 - accur
acy: 0.8438 - val_loss: 0.7750 - val_accuracy: 0.6250
Epoch 49/50
1/1 [==============================] - 17s 17s/step - loss: 0.2443 - accur
acy: 0.9375 - val_loss: 0.7650 - val_accuracy: 0.6875
Epoch 50/50
1/1 [==============================] - 18s 18s/step - loss: 0.2714 - accur
acy: 0.8750 - val_loss: 0.8906 - val_accuracy: 0.6562
```

In [26]:

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

```
1/1 [==============================] - 8s 8s/step - loss: 0.6951 - accuracy:
0.7188
```

In [27]:

```
scores
```

Out[27]:

```
[0.6950811147689819, 0.71875]
```

In [28]:

```
history.history.keys()
```

Out[28]:

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

In [29]:

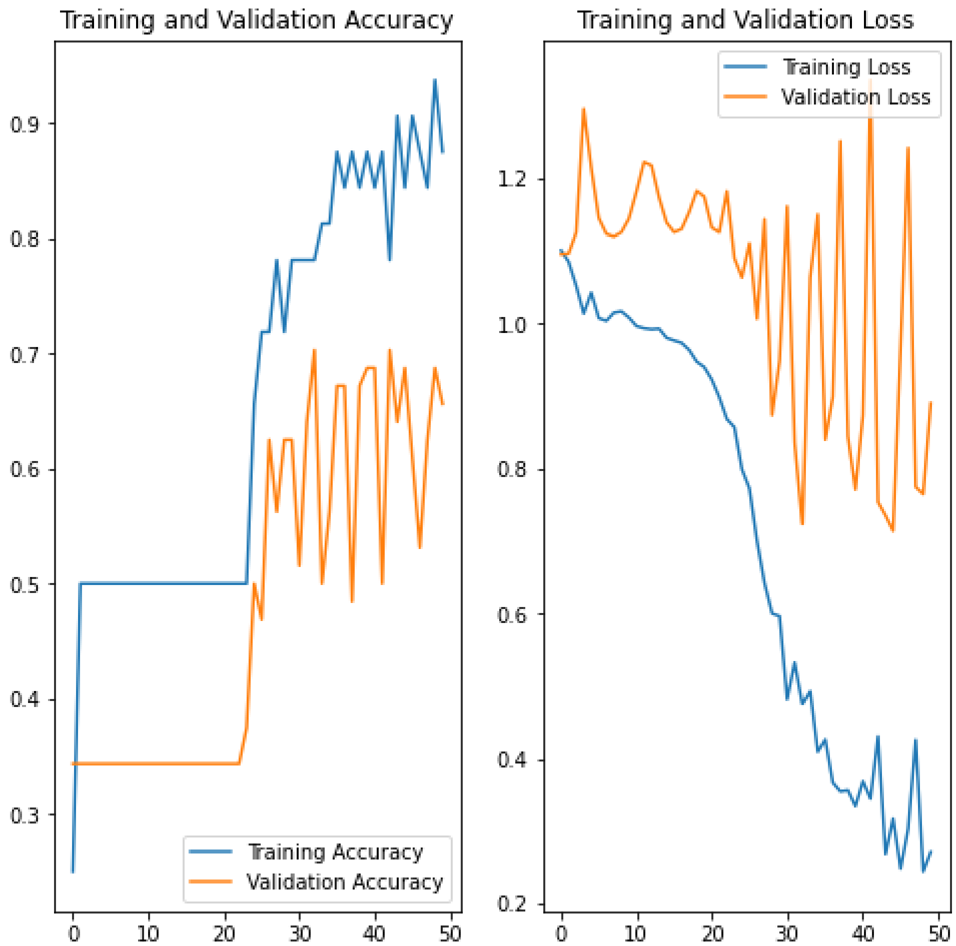
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

In [30]:

```
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()
```



In [31]:

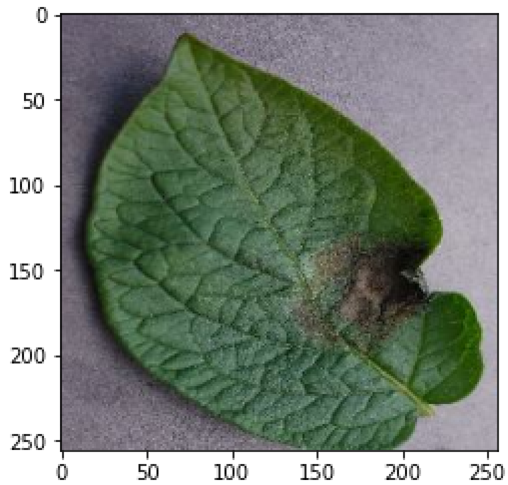
```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    first_image = image_batch[i].numpy().astype('uint8')
    first_label = labels_batch[i].numpy()

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

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

```
first image to predict
actual label: Potato___healthy
predicted label: Potato___Late_blight
```



In [32]:

```
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
```

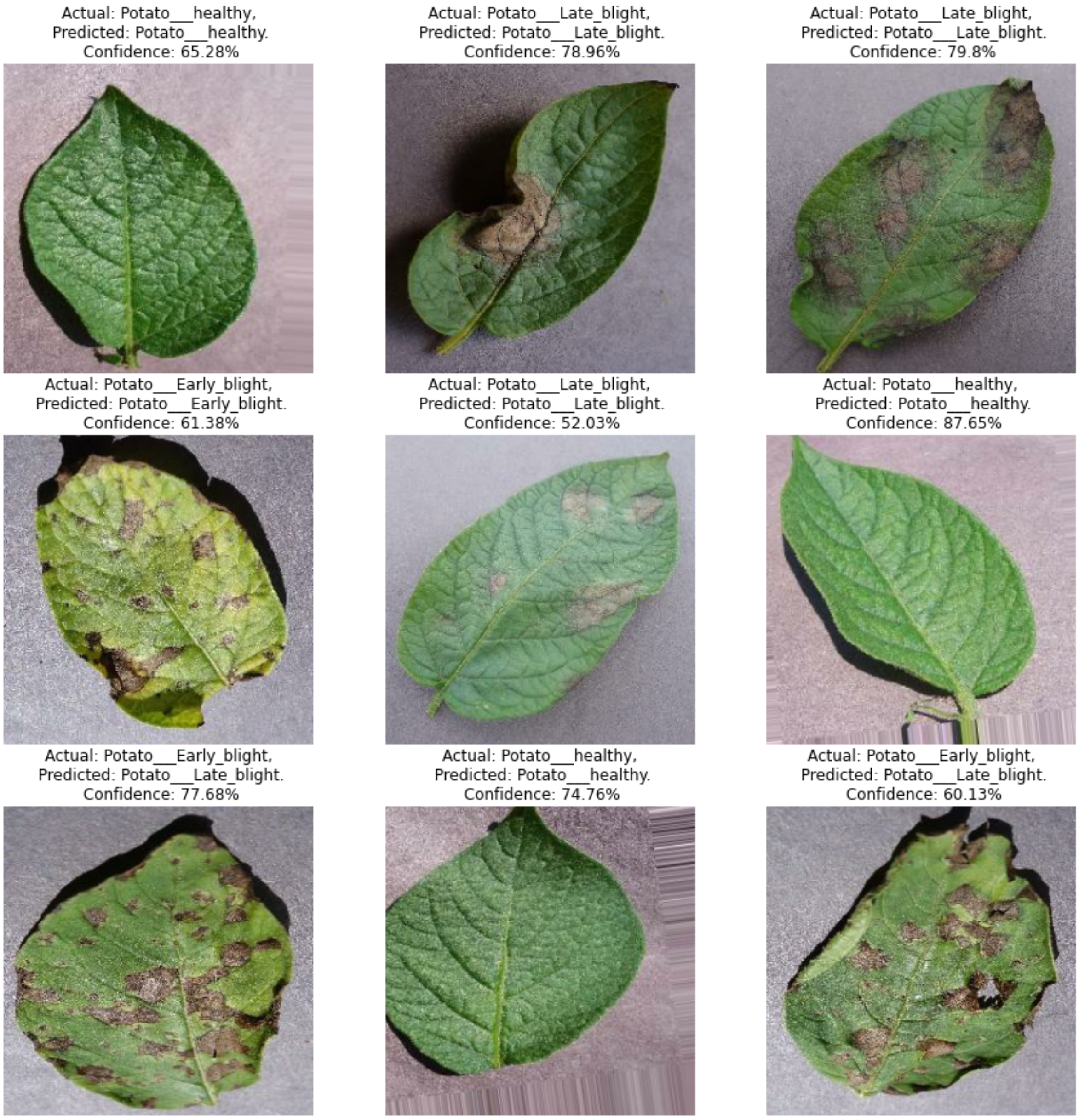
In [33]:

```
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 Confidence: {

        plt.axis("off")
```



In [34]:

```
model.save("potatoes.h5")
```

In [ ]:

```

```