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

In [2]:
```
IMAGE_SIZE=256
BATCH_SIZE=32
CHANNELS=3
EPCOHS=30
```

In [3]:
```
dataset=tf.keras.preprocessing.image_dataset_from_directory(
    "PlantVillage",
    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]:
```
len(dataset) #shows 68 bcz every element in dataset is a batch of 32 images
```

Out[5]:
```
68
```

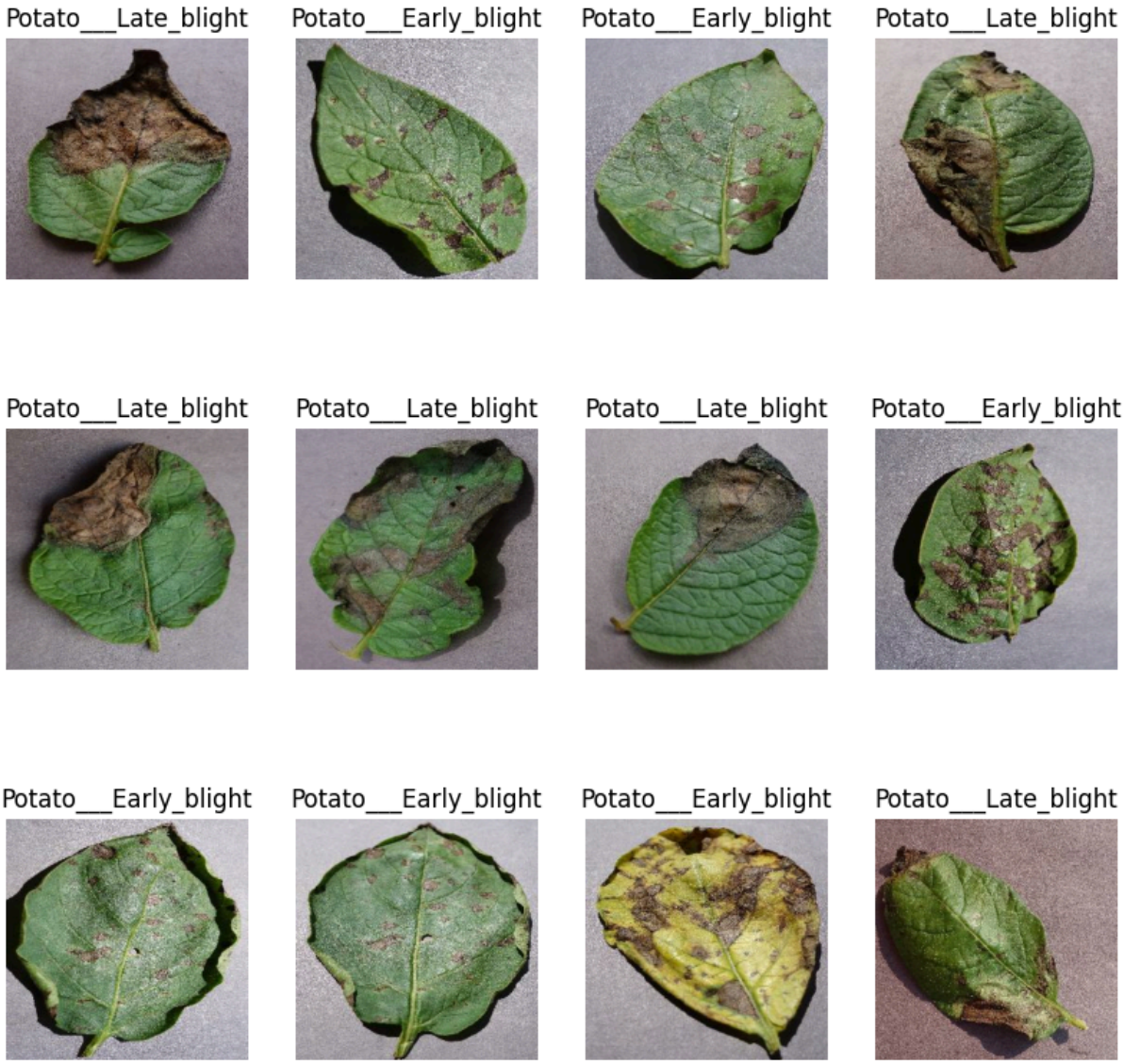
In [6]:
```
68*32 #Last batch not perfect so shows a bit more than that, 68 BATCHES OF 32 IMAGE
```

Out[6]:
```
2176
```

In [7]:
```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

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

In [8]:
```
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")
```



In [ ]:
```
Function to Split Dataset
Dataset should be bifurcated into 3 subsets, namely:

    1.Training: Dataset to be used while training 80%
    2.Validation: Dataset to be tested against while training 10%
    3.Test: Dataset to be tested against after we trained a model 10%
```

In [9]:
```
train_size=0.8
len(dataset)*train_size
```

Out[9]: 54.400000000000006

In [10]:
```
train_ds=dataset.take(54)
len(train_ds)
```

Out[10]: 54

In [11]:
```
test_ds=dataset.skip(54)
len(test_ds)
```

Out[11]: 14

In [12]:
```
val_size=0.1
len(dataset)*val_size
```

Out[12]: 6.800000000000001

In [13]:
```
val_ds=test_ds.take(6)
len(val_ds)
```

Out[13]: 6

In [14]:
```
test_ds = test_ds.skip(6)
len(test_ds)
```

Out[14]: 8

In [15]:
```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, s
    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 [16]:
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

In [17]:
```
len(train_ds)
```

Out[17]: 54

In [18]:
```
len(val_ds)
```

Out[18]: 6

In [19]:
```
len(test_ds)
```

Out[19]: 8

In [ ]:
```
Cache, Shuffle, and Prefetch the Dataset #Improves performance by reading and keepi
```

In [20]:
```
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 [21]:
```
resize_and_rescale = tf.keras.Sequential([
  layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.Rescaling(1.0/255),
])
```

In [ ]:
```
Data Augmentation
```

In [22]:
```
data_augmentation = tf.keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"),
  layers.RandomRotation(0.2),
])
```

In [ ]:
```
Model Architecture || Building model
We use a CNN coupled with a Softmax activation in the output layer.
We also add the initial layers for resizing, normalization and Data Augmentation.
```

In [23]:
```
model = models.Sequential([
    layers.Input(shape=(IMAGE_SIZE, IMAGE_SIZE, CHANNELS)),
    resize_and_rescale,

    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, kernel_size=(3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(3, activation='softmax')
])
```

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

Model: "sequential_2"

Layer (type)	Output Shape	
sequential (Sequential)	(None, 256, 256, 3)	
conv2d (Conv2D)	(None, 254, 254, 32)	
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	
conv2d_1 (Conv2D)	(None, 125, 125, 64)	
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	
conv2d_2 (Conv2D)	(None, 60, 60, 64)	
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 64)	
conv2d_3 (Conv2D)	(None, 28, 28, 64)	
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 64)	
flatten (Flatten)	(None, 12544)	
dense (Dense)	(None, 64)	
dense_1 (Dense)	(None, 3)	

Total params: 896,323 (3.42 MB)

Trainable params: 896,323 (3.42 MB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
Compiling the Model
We use adam Optimizer, SparseCategoricalCrossentropy for losses, accuracy as a metr
```

In [25]:
```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

In [26]:
```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=30,
)
```

```
Epoch 1/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 96s 2s/step - accuracy: 0.5696 - loss: 0.8409 - val_accur
acy: 0.8594 - val_loss: 0.4096
Epoch 2/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 0.8389 - loss: 0.4056 - val_accur
acy: 0.8906 - val_loss: 0.2759
Epoch 3/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 75s 1s/step - accuracy: 0.8897 - loss: 0.2790 - val_accur
acy: 0.9427 - val_loss: 0.1706
Epoch 4/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 0.9405 - loss: 0.1579 - val_accur
acy: 0.9427 - val_loss: 0.1440
Epoch 5/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 0.9587 - loss: 0.1187 - val_accur
acy: 0.8854 - val_loss: 0.2784
Epoch 6/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 69s 1s/step - accuracy: 0.9545 - loss: 0.1414 - val_accur
acy: 0.9844 - val_loss: 0.0720
Epoch 7/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 0.9708 - loss: 0.0963 - val_accur
acy: 0.9531 - val_loss: 0.1308
Epoch 8/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 69s 1s/step - accuracy: 0.9768 - loss: 0.0625 - val_accur
acy: 0.9479 - val_loss: 0.1837
Epoch 9/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 69s 1s/step - accuracy: 0.9611 - loss: 0.1075 - val_accur
acy: 0.9792 - val_loss: 0.0729
Epoch 10/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 0.9658 - loss: 0.0926 - val_accur
acy: 0.9844 - val_loss: 0.0578
Epoch 11/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 0.9903 - loss: 0.0401 - val_accur
acy: 0.9896 - val_loss: 0.0430
Epoch 12/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 0.9869 - loss: 0.0451 - val_accur
acy: 0.9792 - val_loss: 0.0616
Epoch 13/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 69s 1s/step - accuracy: 0.9870 - loss: 0.0270 - val_accur
acy: 0.9844 - val_loss: 0.0419
Epoch 14/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 69s 1s/step - accuracy: 0.9978 - loss: 0.0123 - val_accur
acy: 0.9844 - val_loss: 0.0370
Epoch 15/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 69s 1s/step - accuracy: 0.9980 - loss: 0.0090 - val_accur
acy: 0.9844 - val_loss: 0.0396
Epoch 16/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 69s 1s/step - accuracy: 0.9973 - loss: 0.0065 - val_accur
acy: 0.9844 - val_loss: 0.0554
Epoch 17/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 0.9935 - loss: 0.0218 - val_accur
acy: 0.9792 - val_loss: 0.0322
Epoch 18/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 0.9848 - loss: 0.0413 - val_accur
acy: 0.9896 - val_loss: 0.0238
Epoch 19/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 0.9982 - loss: 0.0098 - val_accur
```

```
acy: 0.9896 - val_loss: 0.0535
Epoch 20/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 1.0000 - loss: 0.0016 - val_accur
acy: 0.9844 - val_loss: 0.0546
Epoch 21/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 69s 1s/step - accuracy: 0.9995 - loss: 0.0020 - val_accur
acy: 0.9896 - val_loss: 0.0365
Epoch 22/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 69s 1s/step - accuracy: 1.0000 - loss: 5.1061e-04 - val_a
ccuracy: 0.9896 - val_loss: 0.0404
Epoch 23/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 1.0000 - loss: 3.3694e-04 - val_a
ccuracy: 0.9896 - val_loss: 0.0462
Epoch 24/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 1.0000 - loss: 2.8329e-04 - val_a
ccuracy: 0.9844 - val_loss: 0.0463
Epoch 25/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 1.0000 - loss: 1.6416e-04 - val_a
ccuracy: 0.9844 - val_loss: 0.0495
Epoch 26/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 1.0000 - loss: 1.5142e-04 - val_a
ccuracy: 0.9844 - val_loss: 0.0508
Epoch 27/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 72s 1s/step - accuracy: 1.0000 - loss: 1.0064e-04 - val_a
ccuracy: 0.9844 - val_loss: 0.0515
Epoch 28/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 73s 1s/step - accuracy: 1.0000 - loss: 9.1430e-05 - val_a
ccuracy: 0.9844 - val_loss: 0.0523
Epoch 29/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 1.0000 - loss: 7.9466e-05 - val_a
ccuracy: 0.9844 - val_loss: 0.0542
Epoch 30/30
54/54 ━━━━━━━━━━━━━━━━━━━━ 70s 1s/step - accuracy: 1.0000 - loss: 8.2960e-05 - val_a
ccuracy: 0.9844 - val_loss: 0.0549
```

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

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 10s 306ms/step - accuracy: 0.9991 - loss: 0.0048
```

In [ ]:
```
Achieved 99.91% accuracy on our test dataset
```

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

Out[28]:
```
[0.02119874581694603, 0.99609375]
```

In [ ]:
```
Plotting the Accuracy and Loss Curves
```

In [29]:
```
history.params
```

Out[29]:
```
{'verbose': 1, 'epochs': 30, 'steps': 54}
```

In [30]:
```
history.history.keys()
```

Out[30]:
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

In [ ]:
```
loss, accuracy, val loss etc are a python list containing values of loss, accuracy
```

In [31]:
```
type(history.history['loss'])
```

Out[31]:
```
list
```

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

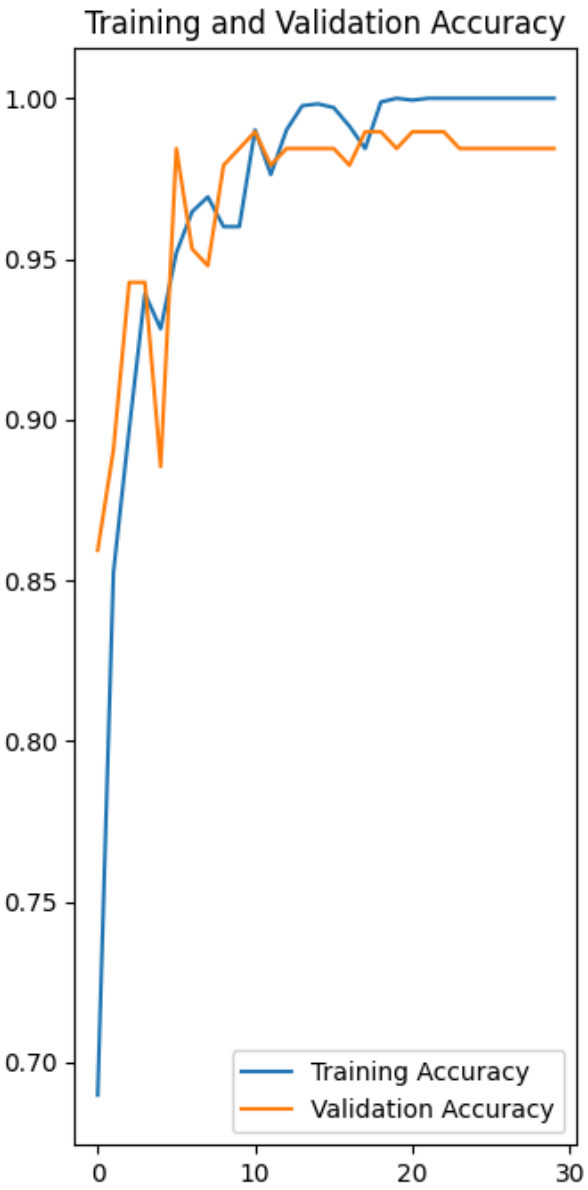
Out[32]:
```
30
```

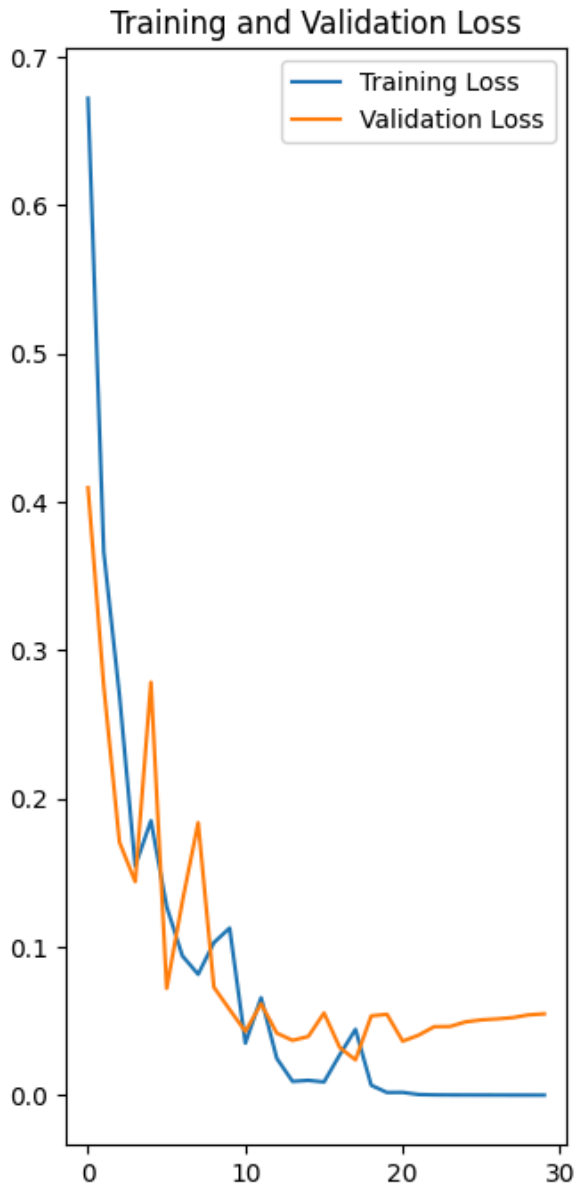
In [33]:
```
history.history['loss'][:5] # show loss for first 5 epochs
```

Out[33]:
```
[0.6720696687698364,
 0.36627545952796936,
 0.2696300745010376,
 0.15447428822517395,
 0.18507708609104156]
```

In [34]:
```
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(30), acc, label='Training Accuracy')
plt.plot(range(30), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(30), loss, label='Training Loss')
plt.plot(range(30), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```

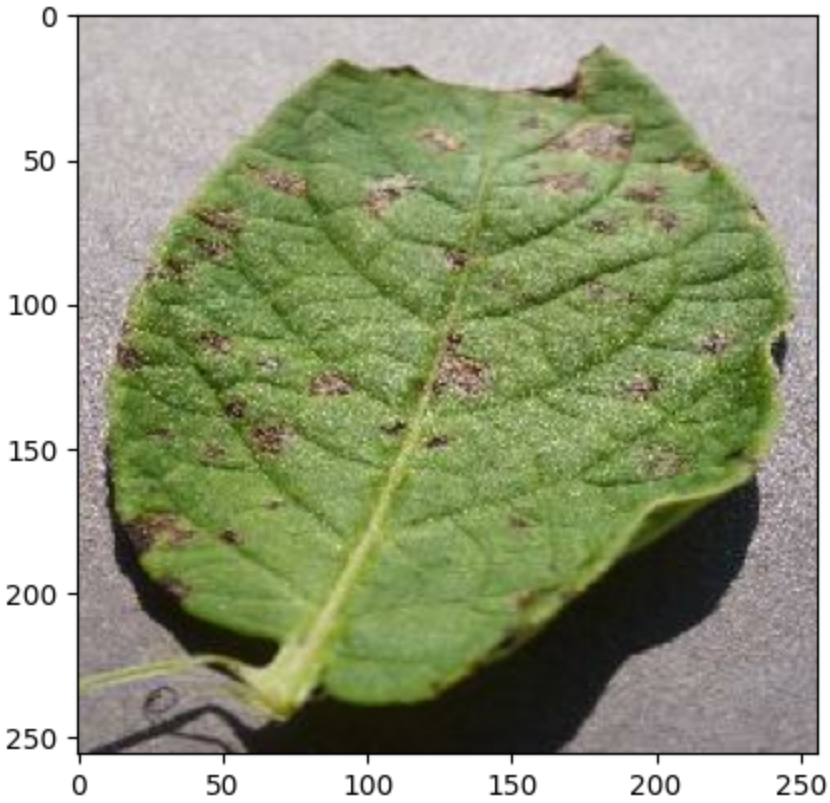




RUNNING PREDICTION ON A SAMPLE IMAGE

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 ━━━━━━━━━━━━━━━━━━━━ 1s 677ms/step
predicted label: Potato___Early_blight
```



In [ ]:
```
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
```

In [ ]:
```
RUNNING INFERENCE ON A 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 Confi

        plt.axis("off")
```

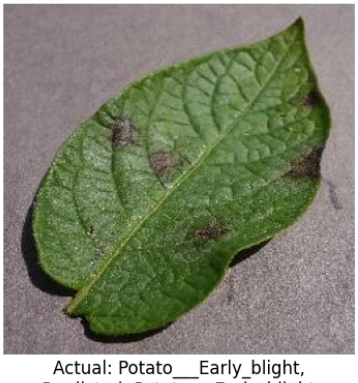
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 340ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 92ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 81ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 71ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 72ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 82ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step
```

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

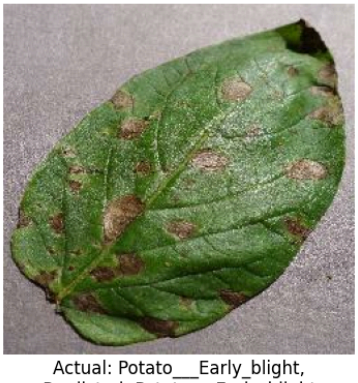
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.98999786376953%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.9000015258789%



Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 99.97000122070312%



Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 100.0%

In [66]:
```
model.save('my_model.keras')
```

In [42]:
```
print(model)
```

```
<Sequential name=sequential_2, built=True>
```

In [40]:
```
import os
```

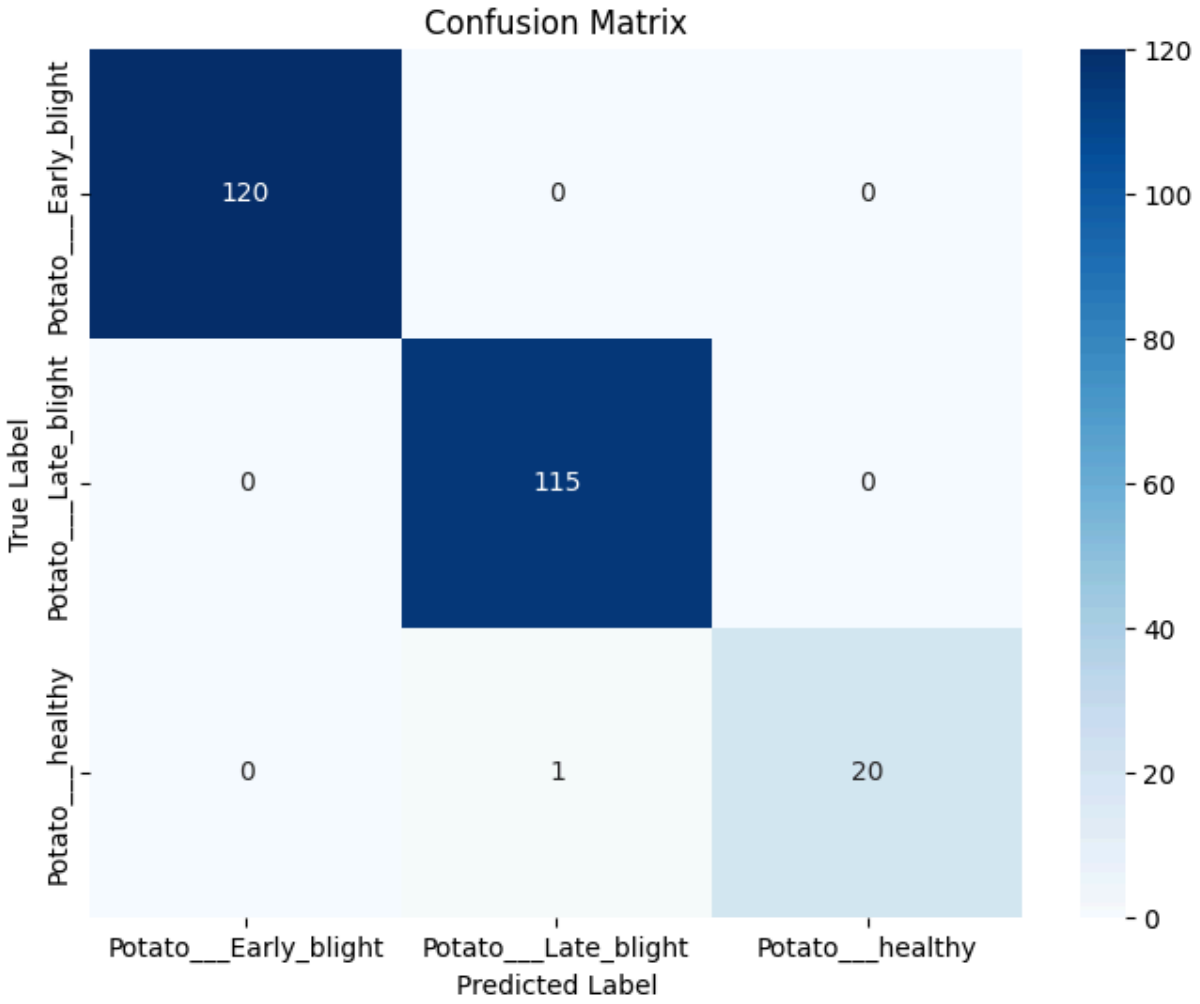
```
models_dir = "../saved_model"

if not os.path.exists(models_dir):
    os.makedirs(models_dir)

model_version = max([int(i.split('.')[0]) for i in os.listdir(models_dir) if i.ends

model.save(f"{models_dir}/model_{model_version}.keras")
```

In [43]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report, recall_score
```

In [44]:
```
y_true = []
y_pred = []

for images, labels in test_ds:
    preds = model.predict(images)
    preds = np.argmax(preds, axis=1)
    y_true.extend(labels.numpy())
    y_pred.extend(preds)

y_true = np.array(y_true)
y_pred = np.array(y_pred)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 528ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 382ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 327ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 372ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 375ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 363ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 357ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 323ms/step
```

In [45]:
```
cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



In [46]:
```
print(classification_report(y_true, y_pred, target_names=class_names))
```

```
                     precision    recall  f1-score   support

Potato___Early_blight       1.00      1.00      1.00       120
 Potato___Late_blight       0.99      1.00      1.00       115
     Potato___healthy       1.00      0.95      0.98        21

             accuracy                           1.00       256
            macro avg       1.00      0.98      0.99       256
         weighted avg       1.00      1.00      1.00       256
```

In [47]:
```
recall_macro = recall_score(y_true, y_pred, average='macro')
recall_per_class = recall_score(y_true, y_pred, average=None)

print(f"Macro Average Recall: {recall_macro:.4f}\n")
for idx, class_name in enumerate(class_names):
    print(f"Recall for {class_name}: {recall_per_class[idx]:.4f}")
```

```
Macro Average Recall: 0.9841

Recall for Potato___Early_blight: 1.0000
Recall for Potato___Late_blight: 1.0000
Recall for Potato___healthy: 0.9524
```

In [49]:
```
model.save('my_model.keras')
```

In [54]:
```
model.save(r"C:\Users\Saad Haris\OneDrive\Desktop\Plant-Disease\saved_model\my_mode
```

In [ ]:
```

```