training

December 21, 2024

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

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

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

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

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

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

[5]:
```
len(dataset)
```

[5]:
```
68
```

[6]:
```
for image_batch,label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
```

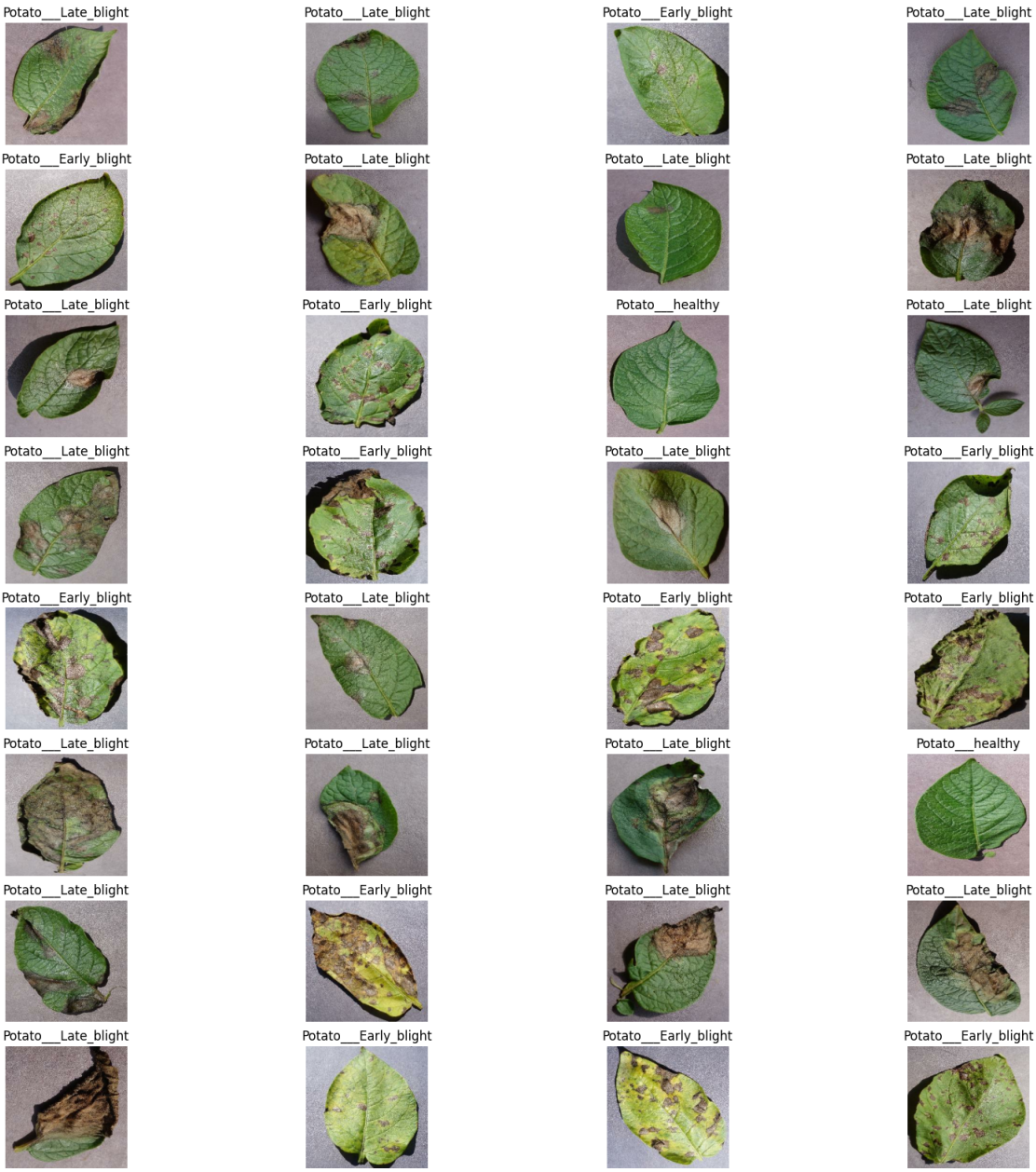
```
(32, 256, 256, 3)
[1 1 1 0 0 0 0 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 0 1 0 0 1 1 2 0 0]
```

[7]:
```
for image_batch,label_batch in dataset.take(1):
    print(image_batch[0].numpy())
```

```
[[[163. 161. 172.]
  [129. 127. 138.]
  [108. 106. 117.]
  …
  [163. 161. 175.]
  [158. 156. 170.]
  [153. 151. 165.]]

 [[149. 147. 158.]
  [ 98.  96. 107.]
  [144. 142. 153.]
  …
  [159. 157. 171.]
  [165. 163. 177.]
  [168. 166. 180.]]

 [[100.  98. 109.]
  [117. 115. 126.]
  [188. 186. 199.]
  …
  [163. 161. 175.]
  [164. 162. 176.]
  [164. 162. 176.]]

 …

 [[142. 138. 153.]
  [120. 116. 131.]
  [136. 132. 147.]
  …
  [180. 178. 191.]
  [178. 176. 189.]
  [189. 187. 200.]]

 [[118. 114. 129.]
  [102.  98. 113.]
  [157. 153. 168.]
  …
  [177. 175. 188.]
  [172. 170. 183.]
  [177. 175. 188.]]

 [[123. 119. 134.]
  [128. 124. 139.]
  [148. 144. 159.]
  …
  [205. 203. 216.]
  [188. 186. 199.]
```

```
[173. 171. 184.]]]
```

[8]:
```
plt.figure(figsize=(20,20))
for image_batch,label_batch in dataset.take(1):
    for i in range(32):
        ax=plt.subplot(8,4,i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        plt.title(class_names[label_batch[i]])
        plt.axis("off")
```



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

[9]:
```
54.400000000000006
```

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

[10]:
```
54
```

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

[11]:
```
14
```

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

[12]:
```
6.800000000000001
```

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

[14]:
```
def get_dataset_partition_tf(ds,train_split=0.8,val_split=0.1,test_split=0.
↪1,shuffle=True,shuffle_size=10000):
    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
```

[15]:
```
train_ds,val_ds,test_ds=get_dataset_partition_tf(dataset)
```

[16]:
```
len(train_ds)+len(test_ds)+len(val_ds)==len(dataset)
```

[16]:
```
True
```

[17]:
```
train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds=test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

[18]:
```
resize_and_rescale=tf.keras.Sequential([
    tf.keras.layers.Resizing(IMAGE_SIZE,IMAGE_SIZE),
    tf.keras.layers.Rescaling(1.0/255)
])
```

[19]:
```
data_augmentation=tf.keras.Sequential([
    tf.keras.layers.RandomFlip("horizontal_and_vertical"),
    tf.keras.layers.RandomRotation(0.2)
])
```

[20]:
```
input_shape=(BATCH_SIZE,IMAGE_SIZE,IMAGE_SIZE,CHANNELS)
n_classes=3

model=models.Sequential([
    resize_and_rescale,
    data_augmentation,
    layers.Conv2D(32,(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)
```

```
C:\Users\admin\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[21]:
```
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)        ␣
↪18,496

max_pooling2d_1 (MaxPooling2D) (32, 62, 62, 64)        ␣
↪  0

conv2d_2 (Conv2D)            (32, 60, 60, 64)          ␣
↪36,928

max_pooling2d_2 (MaxPooling2D) (32, 30, 30, 64)        ␣
↪  0

conv2d_3 (Conv2D)            (32, 28, 28, 64)          ␣
↪36,928

max_pooling2d_3 (MaxPooling2D) (32, 14, 14, 64)        ␣
↪  0

conv2d_4 (Conv2D)            (32, 12, 12, 64)          ␣
↪36,928

max_pooling2d_4 (MaxPooling2D) (32, 6, 6, 64)          ␣
↪  0

conv2d_5 (Conv2D)            (32, 4, 4, 64)            ␣
↪36,928

max_pooling2d_5 (MaxPooling2D) (32, 2, 2, 64)          ␣
↪  0

flatten (Flatten)            (32, 256)                 ␣
↪  0
```

```
dense (Dense)                        (32, 64)                    ␣
↪16,448

dense_1 (Dense)                      (32, 3)                     ␣
↪195
```

Total params: 183,747 (717.76 KB)

Trainable params: 183,747 (717.76 KB)

Non-trainable params: 0 (0.00 B)

[22]:
```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

[23]:
```
history=model.fit(
    train_ds,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    verbose=1,
    validation_data=val_ds
)
```

```
Epoch 1/50
54/54             29s 450ms/step -
accuracy: 0.4653 - loss: 0.9384 - val_accuracy: 0.4688 - val_loss: 0.8447
Epoch 2/50
54/54             24s 431ms/step -
accuracy: 0.6019 - loss: 0.7472 - val_accuracy: 0.7024 - val_loss: 0.6577
Epoch 3/50
54/54             24s 435ms/step -
accuracy: 0.7449 - loss: 0.5253 - val_accuracy: 0.8333 - val_loss: 0.4205
Epoch 4/50
54/54             24s 440ms/step -
accuracy: 0.8216 - loss: 0.4386 - val_accuracy: 0.8594 - val_loss: 0.3878
Epoch 5/50
54/54             24s 438ms/step -
accuracy: 0.8584 - loss: 0.3411 - val_accuracy: 0.8958 - val_loss: 0.2433
Epoch 6/50
54/54             26s 468ms/step -
```

```
accuracy: 0.8727 - loss: 0.3104 - val_accuracy: 0.9010 - val_loss: 0.2243
Epoch 7/50
54/54            25s 456ms/step -
accuracy: 0.8997 - loss: 0.2387 - val_accuracy: 0.9010 - val_loss: 0.2646
Epoch 8/50
54/54            28s 510ms/step -
accuracy: 0.9160 - loss: 0.2120 - val_accuracy: 0.9479 - val_loss: 0.1406
Epoch 9/50
54/54            25s 457ms/step -
accuracy: 0.9363 - loss: 0.1670 - val_accuracy: 0.9531 - val_loss: 0.1482
Epoch 10/50
54/54            24s 439ms/step -
accuracy: 0.9248 - loss: 0.1949 - val_accuracy: 0.8542 - val_loss: 0.4634
Epoch 11/50
54/54            32s 577ms/step -
accuracy: 0.9247 - loss: 0.1809 - val_accuracy: 0.8646 - val_loss: 0.3378
Epoch 12/50
54/54            48s 873ms/step -
accuracy: 0.9493 - loss: 0.1190 - val_accuracy: 0.8594 - val_loss: 0.5386
Epoch 13/50
54/54            48s 870ms/step -
accuracy: 0.9679 - loss: 0.1004 - val_accuracy: 0.9531 - val_loss: 0.1213
Epoch 14/50
54/54            51s 919ms/step -
accuracy: 0.9770 - loss: 0.0672 - val_accuracy: 0.9323 - val_loss: 0.1481
Epoch 15/50
54/54            51s 925ms/step -
accuracy: 0.9837 - loss: 0.0475 - val_accuracy: 0.9427 - val_loss: 0.1685
Epoch 16/50
54/54            52s 934ms/step -
accuracy: 0.9771 - loss: 0.0664 - val_accuracy: 0.9427 - val_loss: 0.1718
Epoch 17/50
54/54            53s 950ms/step -
accuracy: 0.9790 - loss: 0.0588 - val_accuracy: 0.9844 - val_loss: 0.0499
Epoch 18/50
54/54            52s 936ms/step -
accuracy: 0.9803 - loss: 0.0390 - val_accuracy: 0.8750 - val_loss: 0.3598
Epoch 19/50
54/54            48s 871ms/step -
accuracy: 0.9700 - loss: 0.0636 - val_accuracy: 0.9010 - val_loss: 0.4647
Epoch 20/50
54/54            49s 884ms/step -
accuracy: 0.9703 - loss: 0.0717 - val_accuracy: 0.9167 - val_loss: 0.2541
Epoch 21/50
54/54            52s 943ms/step -
accuracy: 0.9775 - loss: 0.0518 - val_accuracy: 0.9740 - val_loss: 0.0493
Epoch 22/50
54/54            53s 959ms/step -
```

```
accuracy: 0.9714 - loss: 0.0571 - val_accuracy: 0.9688 - val_loss: 0.0714
Epoch 23/50
54/54            52s 927ms/step -
accuracy: 0.9944 - loss: 0.0188 - val_accuracy: 0.9062 - val_loss: 0.2783
Epoch 24/50
54/54            50s 907ms/step -
accuracy: 0.9841 - loss: 0.0348 - val_accuracy: 0.9010 - val_loss: 0.2497
Epoch 25/50
54/54            53s 965ms/step -
accuracy: 0.9683 - loss: 0.0793 - val_accuracy: 0.9375 - val_loss: 0.2084
Epoch 26/50
54/54            51s 923ms/step -
accuracy: 0.9794 - loss: 0.0631 - val_accuracy: 0.9740 - val_loss: 0.0901
Epoch 27/50
54/54            50s 902ms/step -
accuracy: 0.9821 - loss: 0.0420 - val_accuracy: 0.9792 - val_loss: 0.0652
Epoch 28/50
54/54            51s 931ms/step -
accuracy: 0.9905 - loss: 0.0367 - val_accuracy: 0.9427 - val_loss: 0.1807
Epoch 29/50
54/54            50s 909ms/step -
accuracy: 0.9938 - loss: 0.0292 - val_accuracy: 0.9896 - val_loss: 0.0310
Epoch 30/50
54/54            51s 926ms/step -
accuracy: 0.9782 - loss: 0.0603 - val_accuracy: 0.9792 - val_loss: 0.0392
Epoch 31/50
54/54            53s 962ms/step -
accuracy: 0.9892 - loss: 0.0288 - val_accuracy: 0.9167 - val_loss: 0.3233
Epoch 32/50
54/54            53s 956ms/step -
accuracy: 0.9761 - loss: 0.0655 - val_accuracy: 0.9792 - val_loss: 0.0465
Epoch 33/50
54/54            51s 937ms/step -
accuracy: 0.9880 - loss: 0.0291 - val_accuracy: 0.9323 - val_loss: 0.2406
Epoch 34/50
54/54            51s 917ms/step -
accuracy: 0.9602 - loss: 0.1119 - val_accuracy: 0.9635 - val_loss: 0.0961
Epoch 35/50
54/54            50s 909ms/step -
accuracy: 0.9918 - loss: 0.0163 - val_accuracy: 0.9940 - val_loss: 0.0116
Epoch 36/50
54/54            50s 907ms/step -
accuracy: 0.9930 - loss: 0.0275 - val_accuracy: 0.9688 - val_loss: 0.0637
Epoch 37/50
54/54            52s 931ms/step -
accuracy: 0.9914 - loss: 0.0249 - val_accuracy: 0.9821 - val_loss: 0.0365
Epoch 38/50
54/54            52s 947ms/step -
```

```
accuracy: 0.9869 - loss: 0.0388 - val_accuracy: 0.9844 - val_loss: 0.0402
Epoch 39/50
54/54             53s 961ms/step -
accuracy: 0.9971 - loss: 0.0148 - val_accuracy: 0.9896 - val_loss: 0.0335
Epoch 40/50
54/54             53s 957ms/step -
accuracy: 0.9957 - loss: 0.0141 - val_accuracy: 0.9583 - val_loss: 0.1145
Epoch 41/50
54/54             51s 921ms/step -
accuracy: 0.9949 - loss: 0.0175 - val_accuracy: 0.9896 - val_loss: 0.0396
Epoch 42/50
54/54             53s 949ms/step -
accuracy: 0.9888 - loss: 0.0313 - val_accuracy: 0.9762 - val_loss: 0.0482
Epoch 43/50
54/54             54s 971ms/step -
accuracy: 0.9925 - loss: 0.0153 - val_accuracy: 0.9896 - val_loss: 0.0388
Epoch 44/50
54/54             54s 973ms/step -
accuracy: 0.9951 - loss: 0.0159 - val_accuracy: 0.9740 - val_loss: 0.0738
Epoch 45/50
54/54             53s 961ms/step -
accuracy: 0.9750 - loss: 0.0802 - val_accuracy: 0.9896 - val_loss: 0.0368
Epoch 46/50
54/54             53s 964ms/step -
accuracy: 0.9951 - loss: 0.0133 - val_accuracy: 0.9948 - val_loss: 0.0129
Epoch 47/50
54/54             53s 964ms/step -
accuracy: 0.9975 - loss: 0.0135 - val_accuracy: 0.9948 - val_loss: 0.0101
Epoch 48/50
54/54             51s 925ms/step -
accuracy: 0.9812 - loss: 0.0401 - val_accuracy: 0.9167 - val_loss: 0.2560
Epoch 49/50
54/54             53s 959ms/step -
accuracy: 0.9932 - loss: 0.0237 - val_accuracy: 0.9688 - val_loss: 0.0741
Epoch 50/50
54/54             53s 960ms/step -
accuracy: 0.9943 - loss: 0.0232 - val_accuracy: 1.0000 - val_loss: 0.0048
```

[24]:
```
scores=model.evaluate(test_ds)
```

```
8/8             4s 261ms/step -
accuracy: 0.9794 - loss: 0.0250
```

[25]:
```
scores
```

[25]:
```
[0.02696279063820839, 0.98828125]
```

[26]:
```
history
```

[26]:
```
<keras.src.callbacks.history.History at 0x18174af9580>
```

[27]:
```
history.params
```

[27]:
```
{'verbose': 1, 'epochs': 50, 'steps': 54}
```

[28]:
```
history.history.keys()
```

[28]:
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

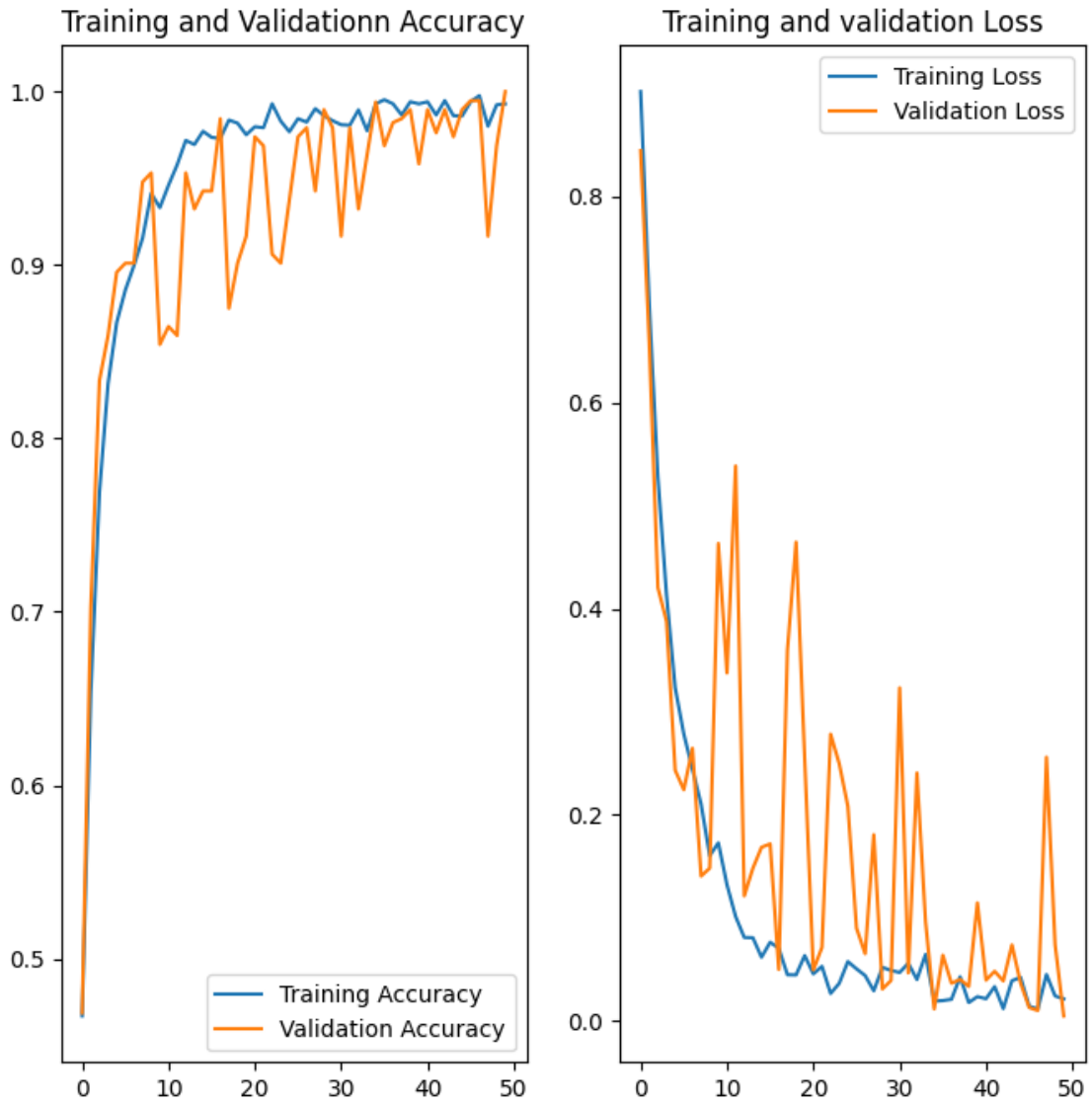
[29]:
```
history.history['accuracy']
```

[29]:
```
[0.4670138955116272,
 0.6531690359115601,
 0.7687793374061584,
 0.8315727710723877,
 0.8667840361595154,
 0.8855633735656738,
 0.8996478915214539,
 0.9154929518699646,
 0.9413145780563354,
 0.9330986142158508,
 0.9460093975067139,
 0.9577465057373047,
 0.9718309640884399,
 0.9694835543632507,
 0.9771126508712769,
 0.9735915660858154,
 0.9730046987533569,
 0.9835680723190308,
 0.9818075299263,
 0.9751157164573669,
 0.9797453880310059,
 0.9791666865348816,
 0.9929577708244324,
 0.9829812049865723,
 0.9768518805503845,
 0.984375,
 0.9823943376541138,
 0.9901620149612427,
 0.9861111044883728,
 0.9832175970077515,
 0.9809027910232544,
 0.9806337952613831,
 0.9894366264343262,
 0.9774305820465088,
 0.9929577708244324,
 0.9953051805496216,
```

```
0.9929577708244324,
0.9865023493766785,
0.9941314458847046,
0.9929577708244324,
0.9941314458847046,
0.9865023493766785,
0.9947916865348816,
0.9861111044883728,
0.98591548204422,
0.9942129850387573,
0.9976525902748108,
0.9800469279289246,
0.9923709034919739,
0.9929577708244324]
```

[30]:
```
acc=history.history['accuracy']
val_acc=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']
```

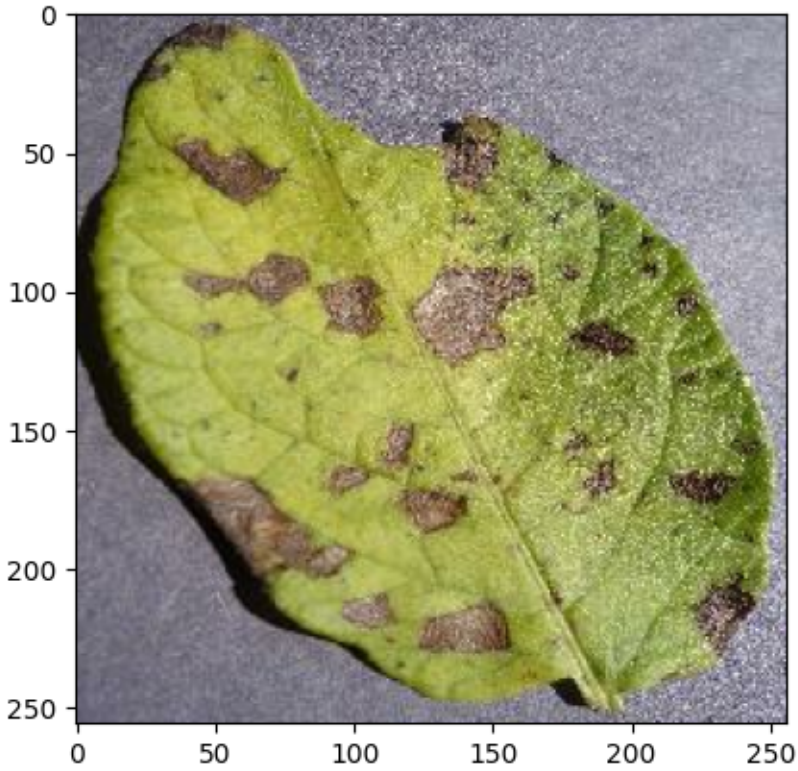
[31]:
```
plt.figure(figsize=(8,8))
plt.subplot(1,2,1)
plt.plot(range(EPOCHS),acc,label='Training Accuracy')
plt.plot(range(EPOCHS),val_acc,label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validationn Accuracy')

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()
```



[32]:
```
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 644ms/step
predicted label: Potato___Early_blight
```

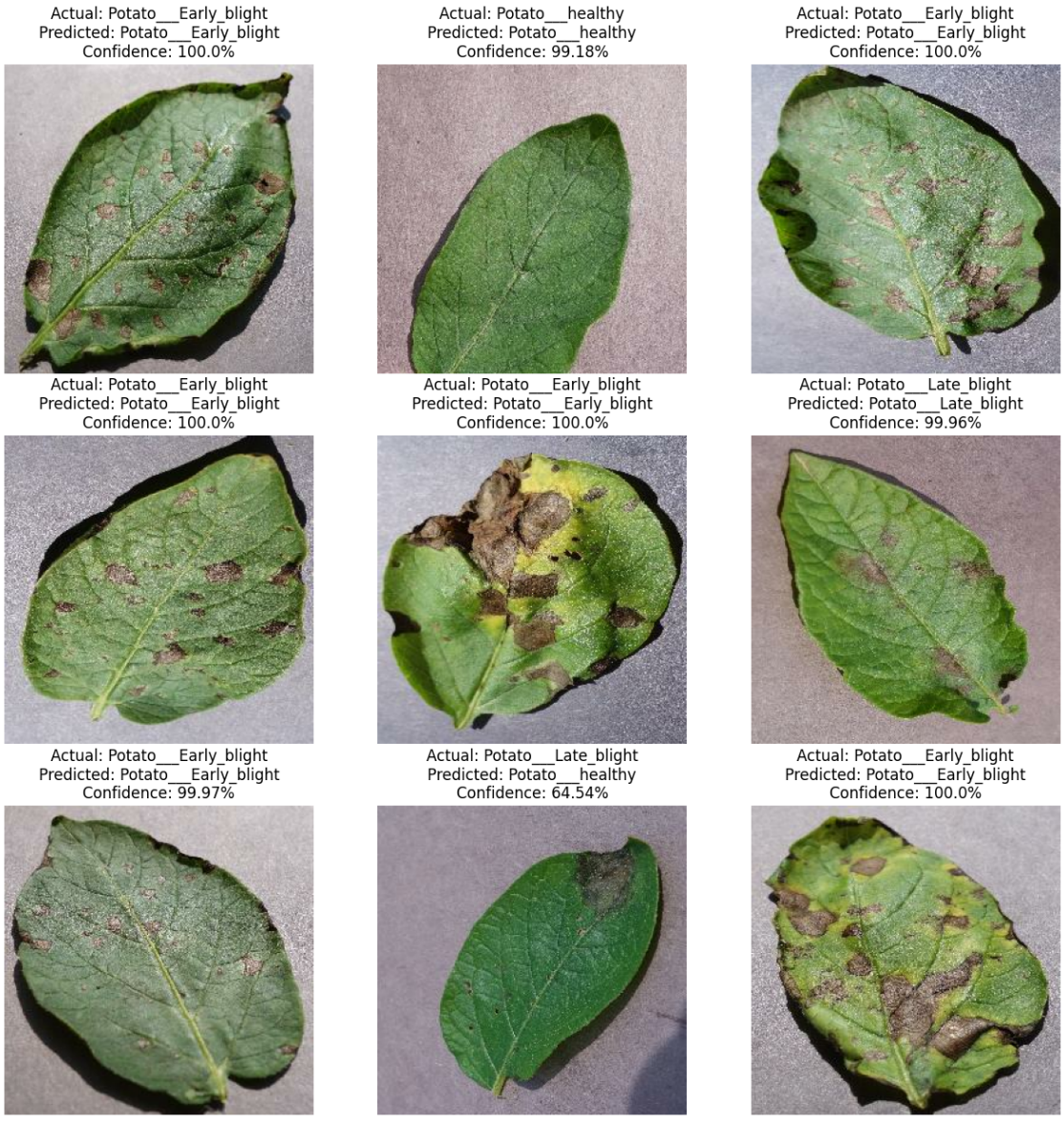


[33]:
```python
def predict(model,img):
    img_array=tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array=tf.expand_dims(img_array,0)

    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
```

[34]:
```python
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}\nPredicted:␣
↪{predicted_class}\nConfidence: {confidence}%")
        plt.axis('off')
```

```
1/1             0s 396ms/step
1/1             0s 106ms/step
1/1             0s 102ms/step
1/1             0s 112ms/step
1/1             0s 98ms/step
1/1             0s 100ms/step
1/1             0s 99ms/step
1/1             0s 102ms/step
1/1             0s 110ms/step
```



[35]:
```
model_version = 1
model.export(f"../models/{model_version}")
```

```
INFO:tensorflow:Assets written to: ../models/1\assets

INFO:tensorflow:Assets written to: ../models/1\assets

Saved artifact at '../models/1'. The following endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 256, 256, 3),
dtype=tf.float32, name='keras_tensor')
Output Type:
  TensorSpec(shape=(None, 3), dtype=tf.float32, name=None)
Captures:
  1655520284304: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520285072: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520287568: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520289104: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520288336: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520290064: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520289680: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520290448: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520288144: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520290832: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520290640: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520291408: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520291600: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520292176: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520292368: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520292944: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520293136: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1655520293712: TensorSpec(shape=(), dtype=tf.resource, name=None)
```