

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=30
```

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)

[2 0 1 2 1 1 1 2 1 2 2 2 2 1 0 1 1 0 0 2 0 1 2 1 0 2 1 1 1 0 0 0]

Potato__healthy



Potato__Early_blight



Potato__Late_blight



Potato__healthy



Potato__Late_blight



Potato__Late_blight



Potato__Late_blight



Potato__healthy



Potato__Late_blight



Potato__healthy



Potato__healthy



Potato__healthy



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=True):
    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)
```

```
[[[0.5411765  0.53333336 0.5882353 ]
  [0.5411765  0.53333336 0.5882353 ]
  [0.54509807 0.5372549  0.5921569 ]
  ...
  [0.4627451  0.43137255 0.48235294]
  [0.56078434 0.5294118  0.5803922 ]
  [0.41568628 0.38431373 0.43529412]]

[[[0.5411765  0.53333336 0.5882353 ]
  [0.5411765  0.53333336 0.5882353 ]
  [0.5372549  0.5294118  0.58431375]
  ...
  [0.56078434 0.5294118  0.5803922 ]
  [0.39215687 0.36078432 0.4117647 ]
  [0.49019608 0.45882353 0.50980395]]

[[[0.5411765  0.53333336 0.5882353 ]
  [0.53333336 0.5254902  0.5803922 ]
  [0.5254902  0.5176471  0.57254905]
  ...
  [0.38039216 0.34901962 0.4          ]
  [0.4627451  0.43137255 0.48235294]
  [0.42745098 0.39607844 0.44705883]]

...

[[[0.6          0.57254905 0.6039216 ]
  [0.60784316 0.5803922  0.6117647 ]
  [0.6156863  0.5882353  0.61960787]
  ...
  [0.42745098 0.38431373 0.40784314]
  [0.47843137 0.43529412 0.45882353]
  [0.43529412 0.39215687 0.41568628]]

[[[0.6          0.57254905 0.6039216 ]
  [0.6156863  0.5882353  0.61960787]
  [0.6313726  0.6039216  0.63529414]
  ...
  [0.3882353  0.34509805 0.36862746]
  [0.42352942 0.38039216 0.40392157]
  [0.3882353  0.34509805 0.36862746]]

[[[0.5803922  0.5529412  0.58431375]
  [0.6039216  0.5764706  0.60784316]
  [0.627451   0.6         0.6313726 ]
  ...
  [0.4117647  0.36862746 0.39215687]
  [0.45490196 0.4117647  0.43529412]
  [0.36078432 0.31764707 0.34117648]]]
```

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 (MaxPooling2D)	(32, 62, 62, 64)	0
conv2d_2 (Conv2D)	(32, 60, 60, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(32, 30, 30, 64)	0
conv2d_3 (Conv2D)	(32, 28, 28, 64)	36928
max_pooling2d_3 (MaxPooling2D)	(32, 14, 14, 64)	0
conv2d_4 (Conv2D)	(32, 12, 12, 64)	36928
max_pooling2d_4 (MaxPooling2D)	(32, 6, 6, 64)	0
conv2d_5 (Conv2D)	(32, 4, 4, 64)	36928
max_pooling2d_5 (MaxPooling2D)	(32, 2, 2, 64)	0
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 1/30

1/1 [=====] - 40s 40s/step - loss: 1.0903 - accuracy: 0.3750 - val_loss: 1.1334 - val_accuracy: 0.3594

Epoch 2/30

1/1 [=====] - 34s 34s/step - loss: 1.0578 - accuracy: 0.3750 - val_loss: 1.2646 - val_accuracy: 0.4375

Epoch 3/30

1/1 [=====] - 14s 14s/step - loss: 1.0661 - accuracy: 0.5625 - val_loss: 1.1776 - val_accuracy: 0.3906

Epoch 4/30

1/1 [=====] - 12s 12s/step - loss: 1.0491 - accuracy: 0.5312 - val_loss: 1.1333 - val_accuracy: 0.2500

Epoch 5/30

1/1 [=====] - 12s 12s/step - loss: 1.0526 - accuracy: 0.4062 - val_loss: 1.1241 - val_accuracy: 0.4844

Epoch 6/30

1/1 [=====] - 13s 13s/step - loss: 1.0532 - accuracy: 0.6250 - val_loss: 1.1304 - val_accuracy: 0.3750

Epoch 7/30

1/1 [=====] - 12s 12s/step - loss: 1.0461 - accuracy: 0.4062 - val_loss: 1.1508 - val_accuracy: 0.3750

Epoch 8/30

1/1 [=====] - 13s 13s/step - loss: 1.0441 - accuracy: 0.4062 - val_loss: 1.1735 - val_accuracy: 0.4688

Epoch 9/30

1/1 [=====] - 13s 13s/step - loss: 1.0407 - accuracy: 0.5938 - val_loss: 1.1797 - val_accuracy: 0.5938

Epoch 10/30

1/1 [=====] - 13s 13s/step - loss: 1.0349 - accuracy: 0.7188 - val_loss: 1.1642 - val_accuracy: 0.5938

Epoch 11/30

1/1 [=====] - 13s 13s/step - loss: 1.0258 - accuracy: 0.7188 - val_loss: 1.1384 - val_accuracy: 0.6094

Epoch 12/30

1/1 [=====] - 13s 13s/step - loss: 1.0123 - accuracy: 0.7500 - val_loss: 1.1178 - val_accuracy: 0.4375

Epoch 13/30

1/1 [=====] - 13s 13s/step - loss: 1.0003 - accuracy: 0.6250 - val_loss: 1.0960 - val_accuracy: 0.5469

Epoch 14/30

1/1 [=====] - 13s 13s/step - loss: 0.9760 - accuracy: 0.7500 - val_loss: 1.0820 - val_accuracy: 0.6094

Epoch 15/30

1/1 [=====] - 13s 13s/step - loss: 0.9439 - accuracy: 0.7500 - val_loss: 1.0952 - val_accuracy: 0.4688

Epoch 16/30

1/1 [=====] - 13s 13s/step - loss: 0.9089 - accuracy: 0.7500 - val_loss: 1.0922 - val_accuracy: 0.4688

Epoch 17/30

1/1 [=====] - 13s 13s/step - loss: 0.9407 - accuracy: 0.5000 - val_loss: 1.1374 - val_accuracy: 0.4531

```
Epoch 18/30
1/1 [=====] - 13s 13s/step - loss: 0.8744 - accur
acy: 0.6562 - val_loss: 0.9981 - val_accuracy: 0.6250
Epoch 19/30
1/1 [=====] - 13s 13s/step - loss: 0.7955 - accur
acy: 0.7500 - val_loss: 0.9291 - val_accuracy: 0.6094
Epoch 20/30
1/1 [=====] - 13s 13s/step - loss: 0.7971 - accur
acy: 0.7188 - val_loss: 0.9856 - val_accuracy: 0.4531
Epoch 21/30
1/1 [=====] - 15s 15s/step - loss: 0.7736 - accur
acy: 0.7188 - val_loss: 0.8563 - val_accuracy: 0.6250
Epoch 22/30
1/1 [=====] - 16s 16s/step - loss: 0.6970 - accur
acy: 0.7500 - val_loss: 0.8314 - val_accuracy: 0.6094
Epoch 23/30
1/1 [=====] - 14s 14s/step - loss: 0.6958 - accur
acy: 0.7188 - val_loss: 1.0167 - val_accuracy: 0.4531
Epoch 24/30
1/1 [=====] - 13s 13s/step - loss: 0.7173 - accur
acy: 0.7500 - val_loss: 0.7829 - val_accuracy: 0.6094
Epoch 25/30
1/1 [=====] - 14s 14s/step - loss: 0.6459 - accur
acy: 0.7188 - val_loss: 0.7636 - val_accuracy: 0.6250
Epoch 26/30
1/1 [=====] - 14s 14s/step - loss: 0.5926 - accur
acy: 0.7500 - val_loss: 0.9457 - val_accuracy: 0.4844
Epoch 27/30
1/1 [=====] - 13s 13s/step - loss: 0.6210 - accur
acy: 0.7500 - val_loss: 0.7309 - val_accuracy: 0.6250
Epoch 28/30
1/1 [=====] - 13s 13s/step - loss: 0.5703 - accur
acy: 0.7500 - val_loss: 0.7221 - val_accuracy: 0.6406
Epoch 29/30
1/1 [=====] - 13s 13s/step - loss: 0.5609 - accur
acy: 0.7188 - val_loss: 0.8552 - val_accuracy: 0.5938
Epoch 30/30
1/1 [=====] - 13s 13s/step - loss: 0.5340 - accur
acy: 0.7500 - val_loss: 0.8829 - val_accuracy: 0.5156
```

In [26]:

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

```
1/1 [=====] - 7s 7s/step - loss: 0.8830 - accuracy:
0.5000
```

In [27]:

```
scores
```

Out[27]:

```
[0.8830219507217407, 0.5]
```

In [28]:

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

Out[28]:

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

In [29]:

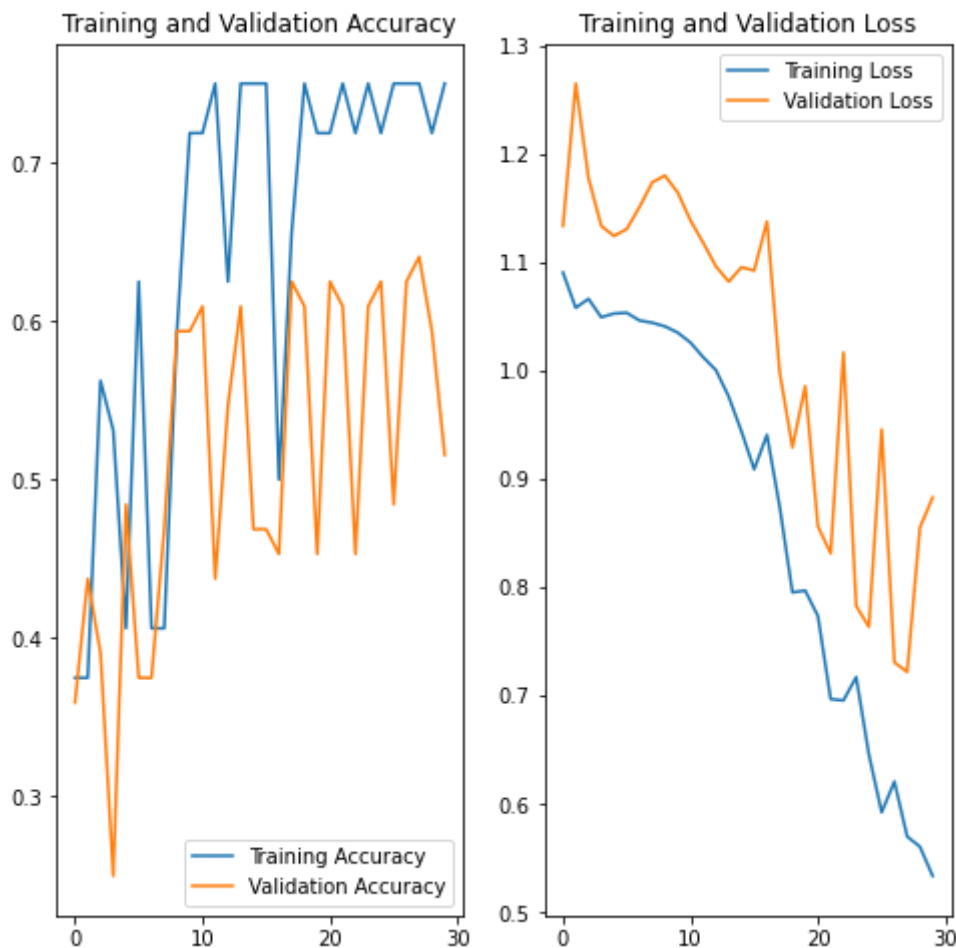
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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]:

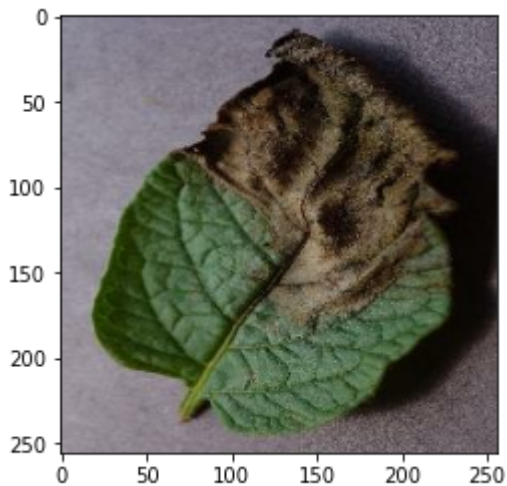
```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    first_image = image_batch[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(image_batch)
    print("predicted label:", class_names[np.argmax(batch_prediction[0])])
```

first image to predict
actual label: Potato__Late_blight
predicted label: Potato__Early_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: {confidence}")

        plt.axis("off")
```

Actual: Potato__Late_blight,
Predicted: Potato__Early_blight.
Confidence: 93.85%



Actual: Potato__Early_blight,
Predicted: Potato__Early_blight.
Confidence: 89.86%



Actual: Potato__healthy,
Predicted: Potato__healthy.
Confidence: 87.92%



Actual: Potato__Early_blight,
Predicted: Potato__Early_blight.
Confidence: 94.41%



Actual: Potato__Late_blight,
Predicted: Potato__Early_blight.
Confidence: 53.9%



Actual: Potato__Early_blight,
Predicted: Potato__Early_blight.
Confidence: 93.57%



Actual: Potato__Late_blight,
Predicted: Potato__Early_blight.
Confidence: 78.92%



Actual: Potato__healthy,
Predicted: Potato__Early_blight.
Confidence: 61.63%



Actual: Potato__Early_blight,
Predicted: Potato__Early_blight.
Confidence: 85.1%



In [34]:

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

In []: