In [4]:
```python
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML
```

In [5]:
```python
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=50
```

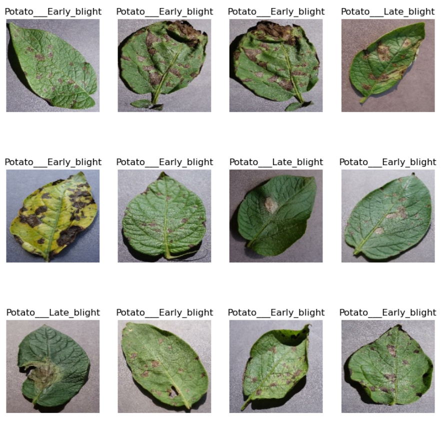
In [9]:
```python
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "Downloads/Plantvillage",
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```
```
Found 2152 files belonging to 4 classes.
```

In [11]:
```python
class_names = dataset.class_names
class_names
```
Out[11]:
```
['.ipynb_checkpoints',
 'Potato___Early_blight',
 'Potato___Late_blight',
 'Potato___healthy']
```

In [12]:
```python
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```
```
(32, 256, 256, 3)
[2 2 2 1 1 1 1 1 2 2 2 2 1 2 1 2 2 2 1 2 1 2 1 1 2 1 1 2 2 3 1 1]
```

In [13]:
```python
#Visualize some of the images from our dataset
```

In [14]:
```python
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")
```


Potato___Early_blight
Potato___Early_blight
Potato___Early_blight
Potato___Late_blight
Potato___Early_blight
Potato___Early_blight
Potato___Late_blight
Potato___Early_blight
Potato___Late_blight
Potato___Early_blight
Potato___Early_blight
Potato___Early_blight

In [15]:
```python
len(dataset)
```
Out[15]:
```
68
```

In [16]:
```python
train_size = 0.8
len(dataset)*train_size
```
Out[16]:
```
54.400000000000006
```

In [17]:
```python
train_ds = dataset.take(54)
len(train_ds)
```
Out[17]:
```
54
```

In [18]:
```python
test_ds = dataset.skip(54)
len(test_ds)
```
Out[18]:
```
14
```

In [19]:
```python
val_size=0.1
len(dataset)*val_size
```
Out[19]:
```
6.800000000000001
```

In [20]:
```python
val_ds = test_ds.take(6)
len(val_ds)
```
Out[20]:
```
6
```

In [21]:
```python
test_ds = test_ds.skip(6)
len(test_ds)
```
Out[21]:
```
8
```

In [22]:
```python
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    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 [23]:
```python
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

In [24]:
```python
len(train_ds)
```
Out[24]:
```
54
```

In [25]:
```python
len(val_ds)
```
Out[25]:
```
6
```

In [26]:
```python
len(test_ds)
```
Out[26]:
```
8
```

In [27]:
```python
#Cache, Shuffle, and Prefetch the Dataset
```

In [28]:
```python
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 [29]:
```python
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

In [30]:
```python
#Data Augmentation
#Data Augmentation is needed when we have less data, this boosts the accuracy of our model by augmenting the data.
```

In [31]:
```python
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```

In [32]:
```python
#Applying Data Augmentation to Train Dataset
```

In [33]:
```python
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```
```
WARNING:tensorflow:From C:\Users\COMP\anaconda3\lib\site-packages\tensorflow\python\autograph\pyct\static_analysis\liveness.py:83: Analyzer.lamba_check (from tensorflow.python.auto
graph.pyct.static_analysis.liveness) is deprecated and will be removed after 2023-09-23.
Instructions for updating:
Lambda fuctions will be no more assumed to be used in the statement where they are used, or at least in the same block. https://github.com/tensorflow/tensorflow/issues/56089
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting RngReadAndSkip cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting Bitcast cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting StatelessRandomUniformV2 cause there is no registered converter for this op.
WARNING:tensorflow:Using a while_loop for converting ImageProjectiveTransformV3 cause there is no registered converter for this op.
```

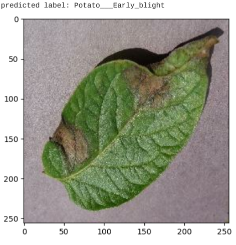
In [34]:
```python
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

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)
```

In [35]:
```python
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 (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 [36]:
```python
#Compiling the Model
```

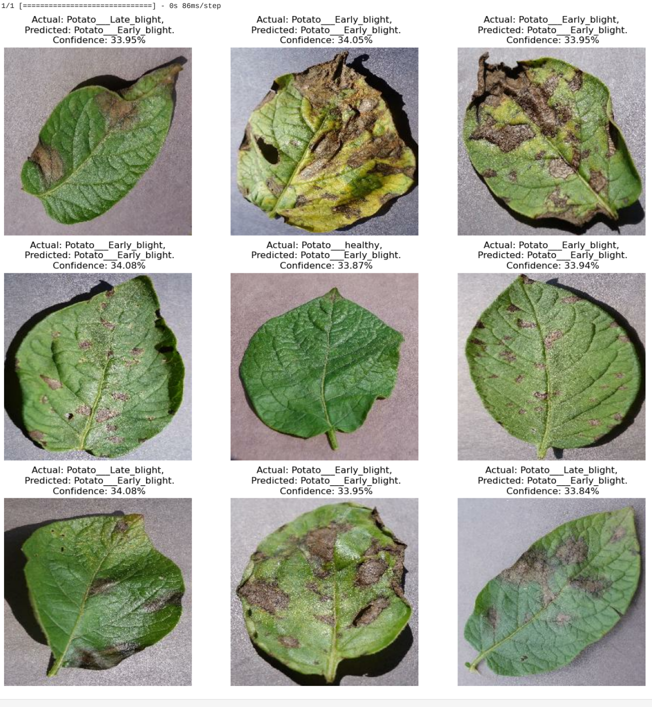
In [37]:
```python
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

In [47]:
```python
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___Late_blight
1/1 [==============================] - 2s 2s/step
predicted label: Potato___Early_blight
```


0
50
100
150
200
250
0
50
100
150
200
250

In [48]:
```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
```

In [49]:
```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},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
```
```
1/1 [==============================] - 0s 343ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 72ms/step
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 95ms/step
1/1 [==============================] - 0s 86ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 86ms/step
```


Actual: Potato___Late_blight,
Predicted: Potato___Early_blight.
Confidence: 33.95%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 34.05%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 33.95%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 34.08%
Actual: Potato___healthy,
Predicted: Potato___Early_blight.
Confidence: 33.87%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 33.94%
Actual: Potato___Late_blight,
Predicted: Potato___Early_blight.
Confidence: 34.08%
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 33.95%
Actual: Potato___Late_blight,
Predicted: Potato___Early_blight.
Confidence: 33.84%

In [ ]: