In [ ]:

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

In [ ]:

```
IMAGE_SIZE = 256
BATCH_SIZE = 32
CHANNELS =3
EPOCHS = 10
```

In [ ]:

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/PlantVillage",
    shuffle = True,
    image_size = (IMAGE_SIZE, IMAGE_SIZE),
    batch_size = BATCH_SIZE
)
```

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

In [ ]:

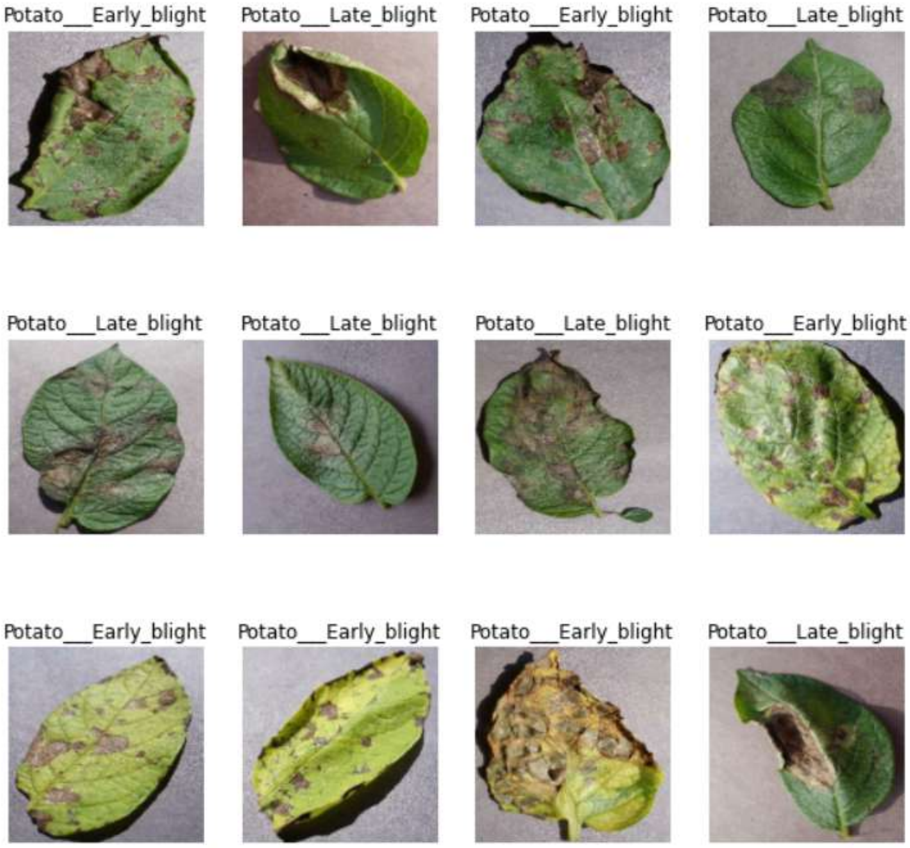
```
class_names = dataset.class_names
class_names
```

Out[ ]:

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

In [ ]:

```
plt.figure(figsize=(10,10))
for image_batch, label_batch in dataset.take(1):
  for i in range(12):
    ax = plt.subplot(3,4,1+i)
    plt.imshow(image_batch[i].numpy().astype("uint8"))
    plt.title(class_names[label_batch[i]])
    plt.axis("off")
```



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

```
Out[ ]: 68
```

In [ ]:

```
train_size = 0.8
len(dataset)*train_size
```

Out[ ]:

```
54.400000000000006
```

In [ ]:

```
train_ds = dataset.take(54)
len(train_ds)
```

Out[ ]:

```
54
```

In [ ]:

```
test_ds = dataset.skip(54)
len(test_ds)
```

Out[ ]:

```
14
```

In [ ]:

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

Out[ ]:

```
6.800000000000001
```

In [ ]:

```
val_ds = test_ds.take(6)
len(val_ds)
```

Out[ ]:

```
6
```

In [ ]:

```
test_ds = test_ds.skip(6)
len(test_ds)
```

Out[ ]:

```
8
```

In [ ]:

```
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)

  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 [ ]:

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

In [ ]:

```
len(train_ds)
```

Out[ ]:

```
54
```

In [ ]:

```
len(val_ds)
```

Out[ ]:

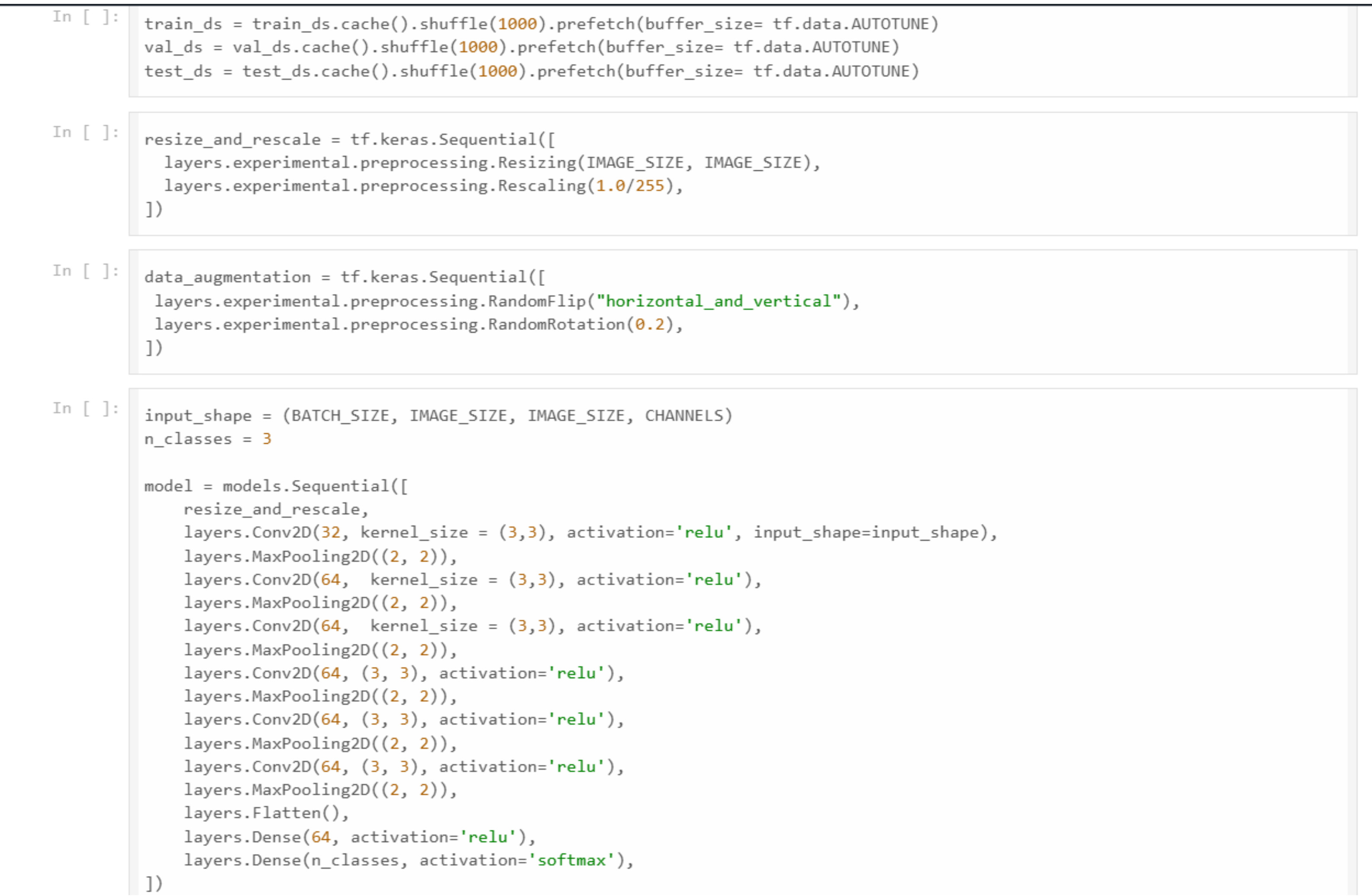
```
6
```

In [ ]:

```
len(test_ds)
```

Out[ ]:

```
8
```



In [ ]:

```
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 [ ]:

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

In [ ]:

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

In [ ]:

```
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 [ ]:

```
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 [ ]:

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

In [ ]:

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

```
Epoch 1/10
54/54 [==============================] - 225s 4s/step - loss: 0.0924 - accuracy: 0.9635 - val_loss: 0.1424 - val_accuracy: 0.
9531
Epoch 2/10
54/54 [==============================] - 207s 4s/step - loss: 0.0974 - accuracy: 0.9653 - val_loss: 0.0619 - val_accuracy: 0.
9688
```

```
Epoch 3/10
54/54 [==============================] - 209s 4s/step - loss: 0.0564 - accuracy: 0.9832 - val_loss: 0.1111 - val_accuracy: 0.
9688
Epoch 4/10
54/54 [==============================] - 217s 4s/step - loss: 0.0284 - accuracy: 0.9884 - val_loss: 0.0508 - val_accuracy: 0.
9688
Epoch 5/10
54/54 [==============================] - 189s 4s/step - loss: 0.0541 - accuracy: 0.9832 - val_loss: 0.0284 - val_accuracy: 0.
9948
Epoch 6/10
54/54 [==============================] - 187s 3s/step - loss: 0.1117 - accuracy: 0.9566 - val_loss: 0.0696 - val_accuracy: 0.
9688
Epoch 7/10
54/54 [==============================] - 185s 3s/step - loss: 0.0297 - accuracy: 0.9925 - val_loss: 0.0634 - val_accuracy: 0.
9792
Epoch 8/10
54/54 [==============================] - 186s 3s/step - loss: 0.0199 - accuracy: 0.9948 - val_loss: 0.0366 - val_accuracy: 0.
9896
Epoch 9/10
54/54 [==============================] - 186s 3s/step - loss: 0.0218 - accuracy: 0.9942 - val_loss: 0.0670 - val_accuracy: 0.
9792
Epoch 10/10
54/54 [==============================] - 180s 3s/step - loss: 0.0121 - accuracy: 0.9983 - val_loss: 0.1130 - val_accuracy: 0.
9635
```

In [ ]:

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

```
8/8 [==============================] - 12s 863ms/step - loss: 0.0094 - accuracy: 0.9961
```

In [ ]:

```
scores
```

Out[ ]:

```
[0.009389366954565048, 0.99609375]
```

In [ ]:

```
history.params
```

```
Out[ ]: {'epochs': 10, 'steps': 54, 'verbose': 1}
```

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

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

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

```
Out[ ]: list
```

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

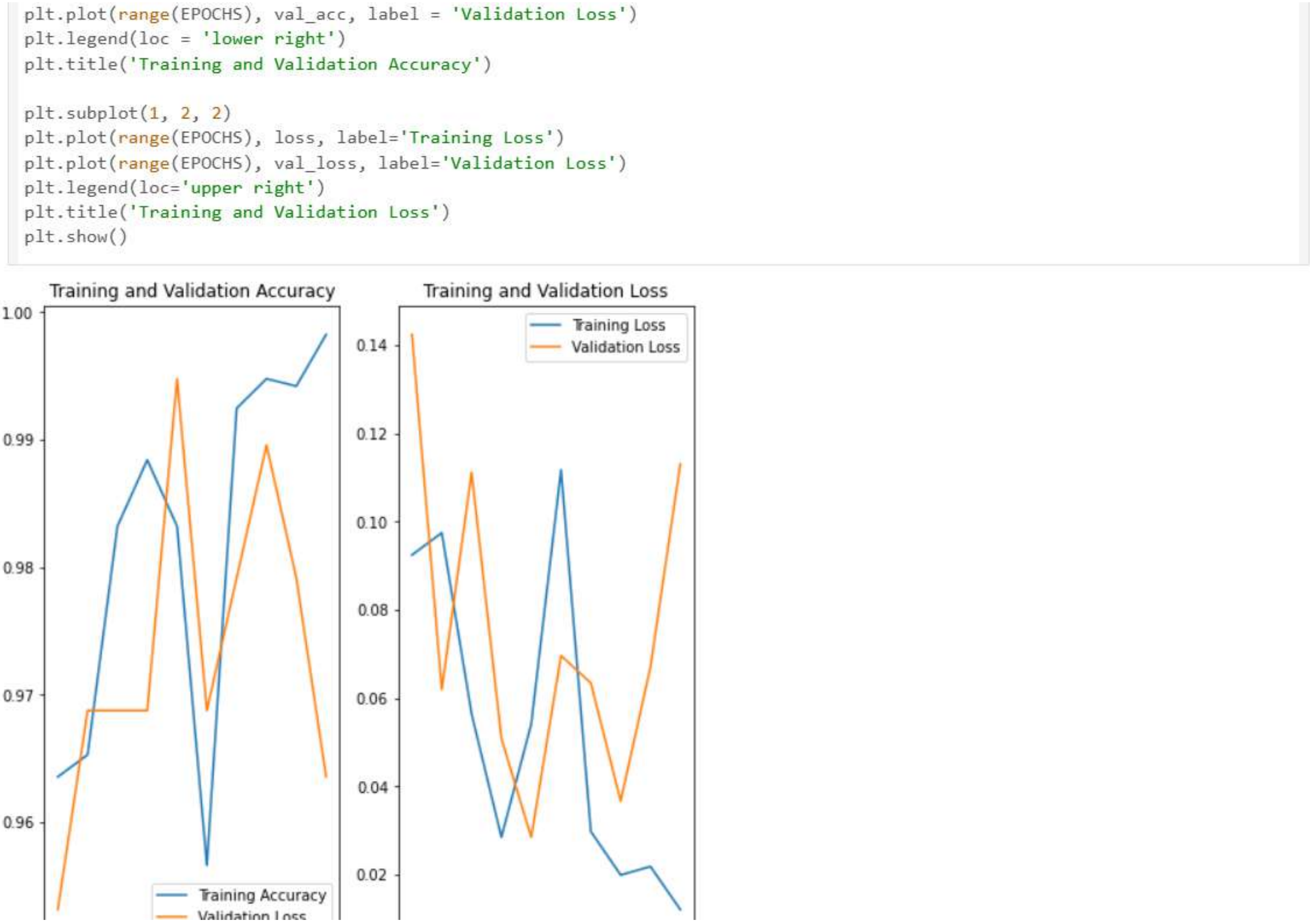
```
Out[ ]: 10
```

```
In [ ]: history.history['loss'][:5]
```

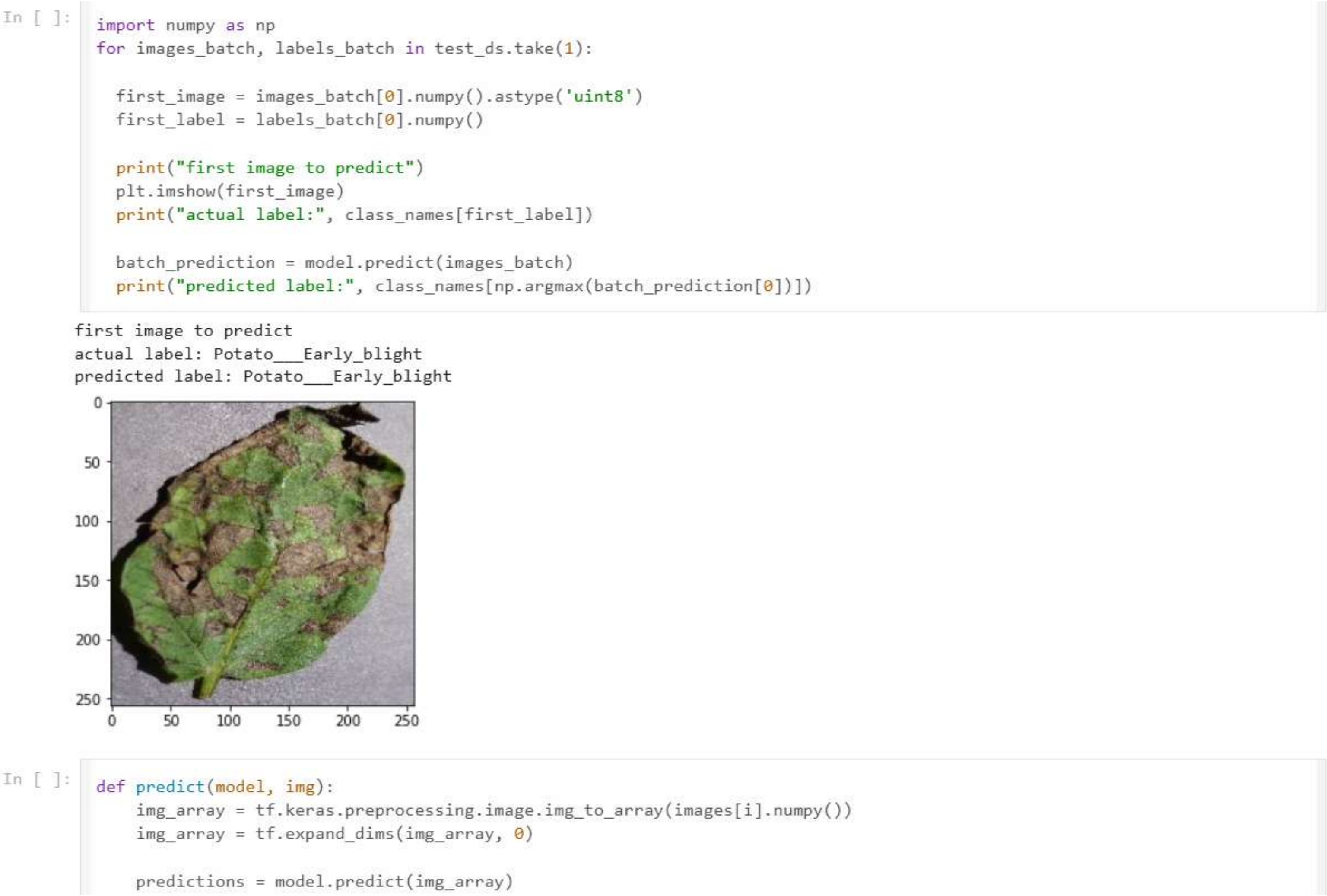
```
Out[ ]: [0.0924125388264656,
         0.09743593633174896,
         0.0563848502933979,
         0.02844615839421749,
         0.05406802520155907]
```

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

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

```
In [ ]: plt.figure(figsize=(8,8))
        plt.subplot(1, 2, 1)
```



```
plt.plot(range(EPOCHS), val_acc, label = 'Validation Loss')
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 [ ]:

```
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])])
```

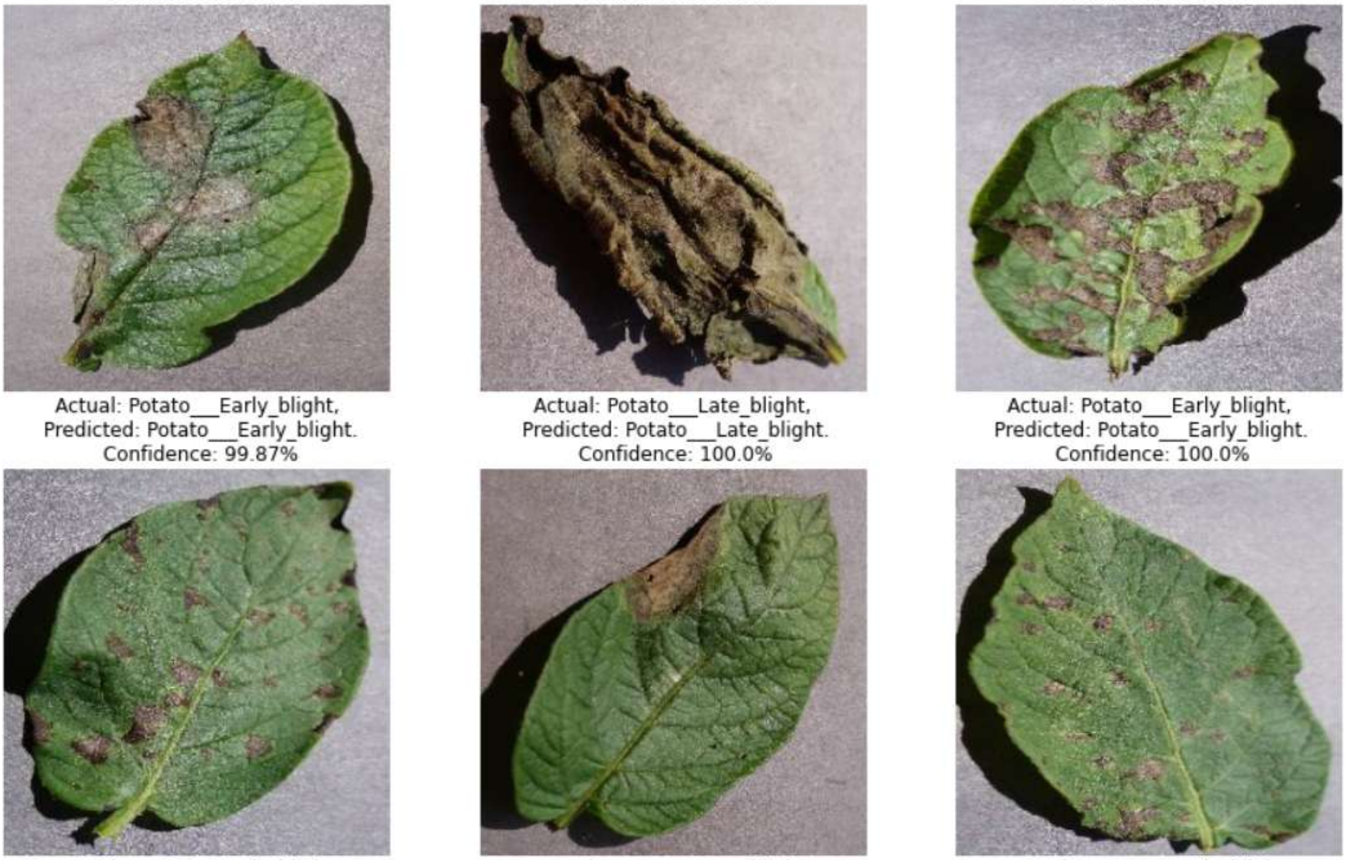
```
first image to predict
actual label: Potato___Early_blight
predicted label: Potato___Early_blight
```

In [ ]:

```
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)
    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 [ ]:

```
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___Late_blight.
Confidence: 100.0%

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

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 99.99%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 99.87%

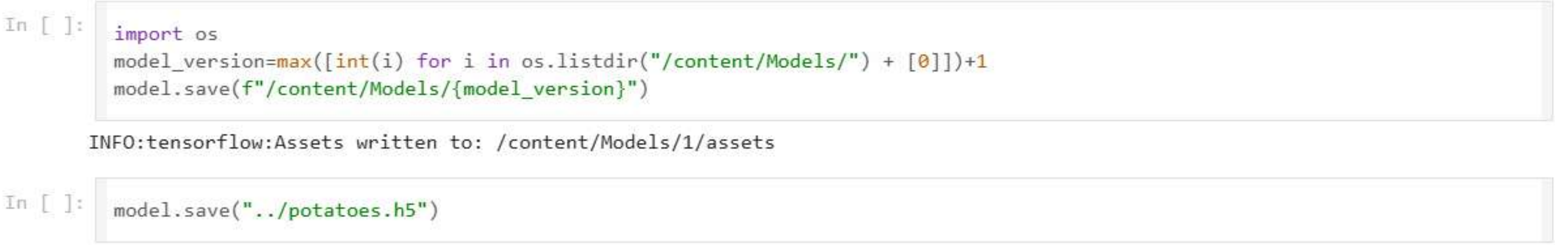
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 100.0%

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

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 99.94%

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

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



```
In [ ]:  import os
         model_version=max([int(i) for i in os.listdir("/content/Models/") + [0]])+1
         model.save(f"/content/Models/{model_version}")

INFO:tensorflow:Assets written to: /content/Models/1/assets

In [ ]:  model.save("../potatoes.h5")
```