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

Double-click (or enter) to edit

```
!pip install rarfile
```

```
Collecting rarfile
  Downloading rarfile-4.2-py3-none-any.whl.metadata (4.4 kB)
Downloading rarfile-4.2-py3-none-any.whl (29 kB)
Installing collected packages: rarfile
Successfully installed rarfile-4.2
```

```
import rarfile
with rarfile.RarFile('/content/PlantVillage.rar', 'r') as rf:
   rf.extractall('/content/drive/MyDrive/Potato_Disease_classification/training')
```

```
#All the constants
BATCH_SIZE = 32 #the number of training samples used in one iteration during the training of
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=5
```

```
#data into tensorflow dataset object

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/Potato_Disease_classification/training/PlantVillage",
    seed=123,
    shuffle=True,#random shuffle
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

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

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

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

```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape) #each batch 32 imgs
    print(labels_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]
```

```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch[0]) #3d array
```

```
tf.Tensor(
[[[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.]
  ...
```

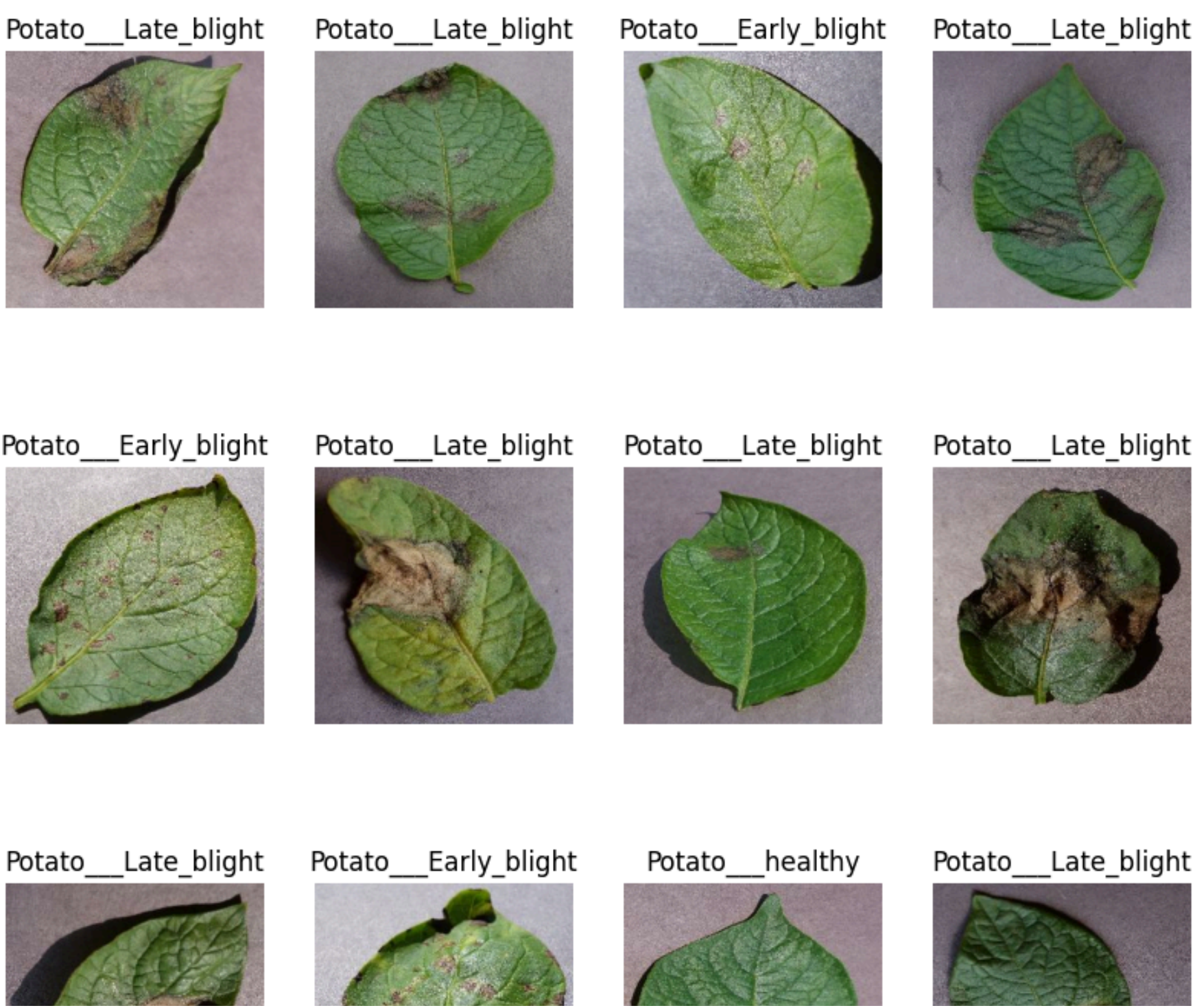
```
        [180. 178. 191.]
        [178. 176. 189.]
        [189. 187. 200.]]

       [[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.]]], shape=(256, 256, 3), dtype=float32)
```

Visualization some of the images in dataset

```
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___Late_blight
Potato___Late_blight
Potato___Early_blight
Potato___Late_blight
Potato___Early_blight
Potato___Late_blight
Potato___Late_blight
Potato___Late_blight
Potato___Late_blight
Potato___Early_blight
Potato___healthy
Potato___Late_blight




```
len(dataset) # 68 batches
```

```
68
```

```
68*32 #total images
```

```
2176
```

Function to Split Dataset

Dataset divided into 3 subsets, namely:

Training: Dataset to be used while training

Validation: Dataset to be tested against while training

Test: Dataset to be tested against after we trained a model

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

```
54.400000000000006
```

```
#for first 54 batches
train_ds = dataset.take(54) #[:54]
len(train_ds)
```

```
54
```

```
test_ds = dataset.skip(54) #[54:]
len(test_ds)
#14batches not for testing divide into val+test dataset
```

```
14
```

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

```
6.800000000000001
```

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

```
6
```

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

```
8
```

```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=T
    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
```

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

```
len(train_ds)
```

```
54
```

```
len(val_ds)
```

```
6
```

```
len(test_ds)
```

```
8
```

Cache, Shuffle, and Prefetch the Dataset #cpu doesnt sit idle even use of gpu

```
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)
```

Building the Model

```
resize_and_rescale = tf.keras.Sequential([
  tf.keras.layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  tf.keras.layers.Rescaling(1./255),
])
```

** Data Augmentation to Train Dataset**

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

```
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Model Architecture

Using CNN coupled with a Softmax activation in the output layer. We also add the initial layers for resizing, normalization and Data Augmentation.

```
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)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
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)	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

Compiling the Model

We use adam Optimizer, SparseCategoricalCrossentropy for losses, accuracy as a metric

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

```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=5,
)
```

```
Epoch 1/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 294s 5s/step - accuracy: 0.4638 - loss: 0.9460 - val_accurac
```

```
Epoch 2/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 259s 5s/step - accuracy: 0.6989 - loss: 0.6786 - val_accurac
Epoch 3/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 264s 5s/step - accuracy: 0.8470 - loss: 0.3562 - val_accurac
Epoch 4/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 300s 6s/step - accuracy: 0.8585 - loss: 0.3227 - val_accurac
Epoch 5/5
54/54 ━━━━━━━━━━━━━━━━━━━━ 268s 5s/step - accuracy: 0.8891 - loss: 0.2610 - val_accurac
```

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

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 19s 1s/step - accuracy: 0.8220 - loss: 0.3878
```

```
scores #[loss,accuracy]
```

```
[0.31998446583747864, 0.8515625]
```

Plotting the Accuracy and Loss Curves

```
history.params
```

```
{'verbose': 1, 'epochs': 5, 'steps': 54}
```

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

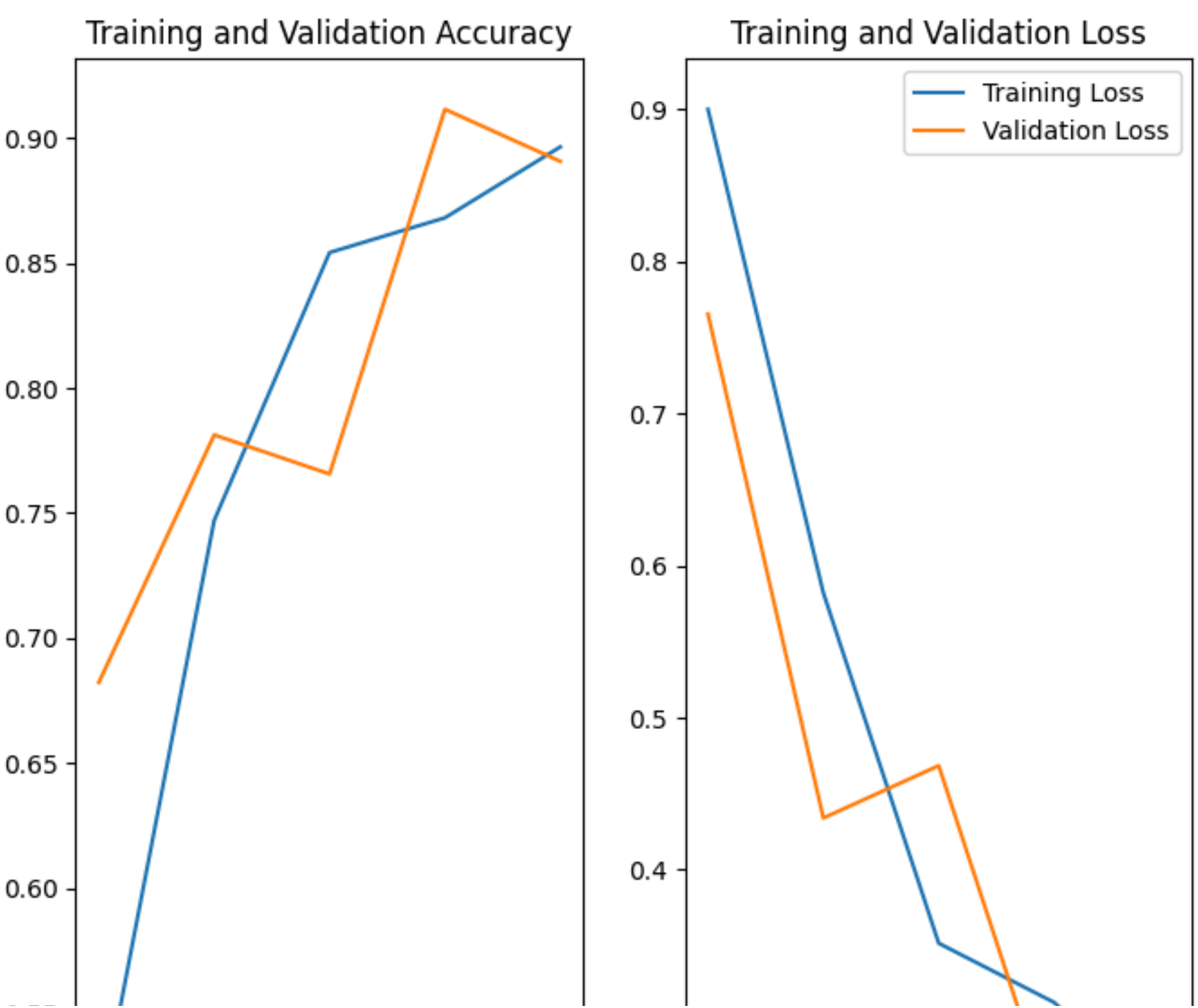
```
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
```

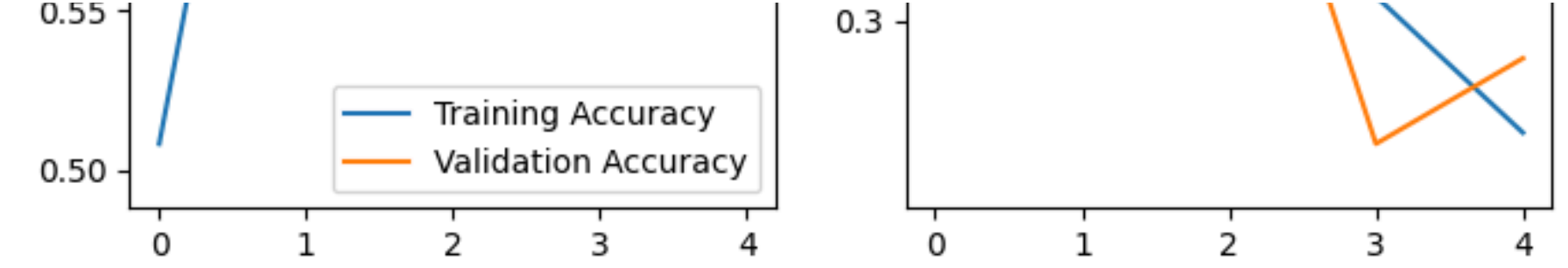
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

```
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()
```


Training and Validation Accuracy
0.90
0.85
0.80
0.75
0.70
0.65
0.60
Training and Validation Loss
Training Loss
Validation Loss
0.9
0.8
0.7
0.6
0.5
0.4



prediction on a sample image

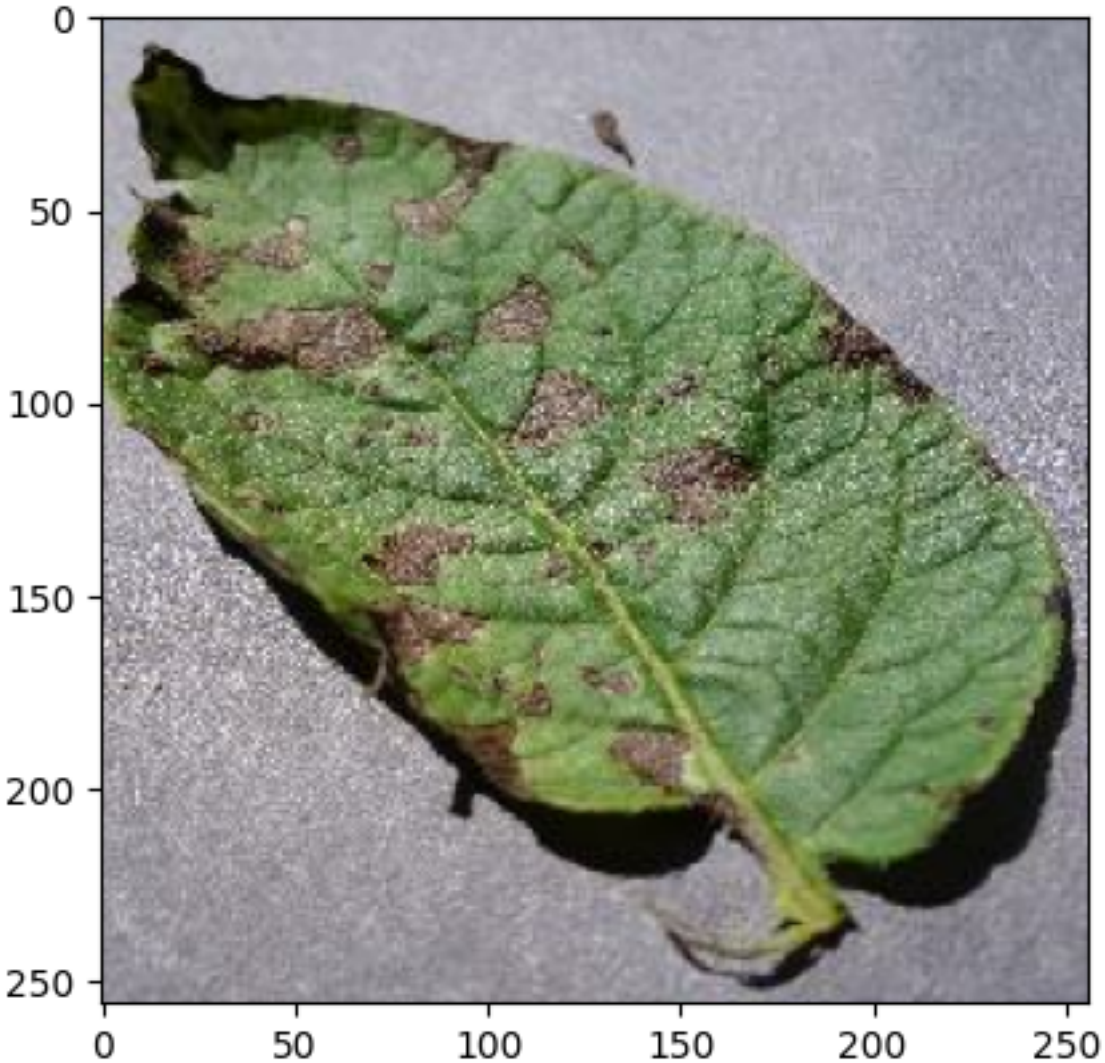
```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    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 1s/step
predicted label: Potato___Early_blight
```



```
#fn for predict
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
```

```
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: {co

        plt.axis("off")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 168ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 54ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 55ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
```

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 64.16%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 99.57%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 67.63%

Actual: Potato___Late_blight,
Predicted: Potato___Early_blight.
Confidence: 77.27%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 98.4%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 98.44%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 91.5%

Actual: Potato___healthy,
Predicted: Potato___healthy.
Confidence: 83.75%

Actual: Potato___Late_blight,
Predicted: Potato___Early_blight.
Confidence: 59.86%