```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import tensorflow as t
from tensorflow.keras import models,layers
import matplotlib.pyplot as plt
import numpy as np
from tensorflow.compat.v1.losses import sparse_softmax_cross_entropy
from tensorflow.compat.v1.ragged import RaggedTensorValue
```

```
IMAGE_SIZE=128
BATCH_SIZE=32
CHANNELS=3
EPOCHS=50 # its a trail and error
```

```
#loading data from google drive
dataset=t.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/Tomato Dataset",
    shuffle=True,
    image_size=( IMAGE_SIZE,IMAGE_SIZE),
     batch_size=BATCH_SIZE
)
```

```
Found 16011 files belonging to 10 classes.
```

```
#set the names according to the classes in Data
class_names = dataset.class_names
class_names
```

```
['Tomato_Bacterial_spot',
 'Tomato_Early_blight',
 'Tomato_Late_blight',
 'Tomato_Leaf_Mold',
 'Tomato_Septoria_leaf_spot',
 'Tomato_Spider_mites_Two_spotted_spider_mite',
 'Tomato__Target_Spot',
 'Tomato__Tomato_YellowLeaf__Curl_Virus',
 'Tomato__Tomato_mosaic_virus',
 'Tomato_healthy']
```

```
# checking the length of Data
#each batch contain 32 images 32*501 =(near to the 16011)
len(dataset)
```

```
501
```

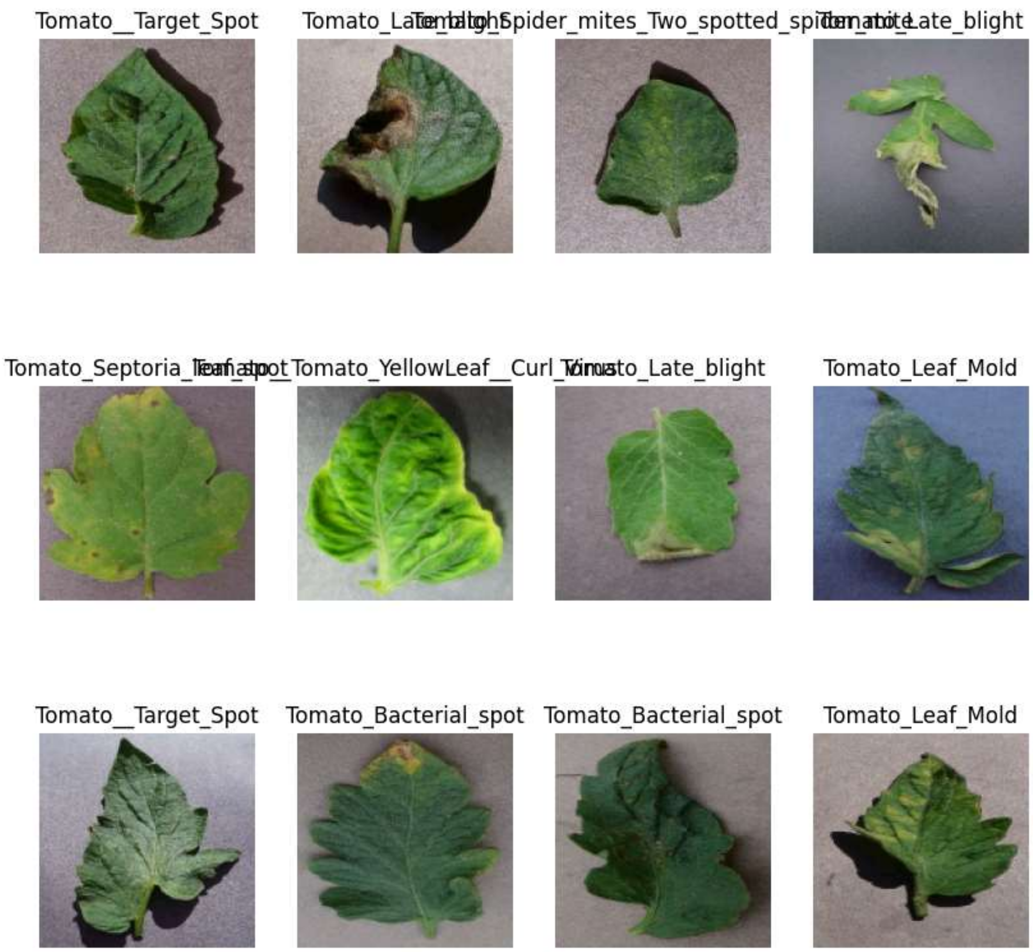
```
#checking the imges
#32 images in 1 batch
#128 x and y
# 3 is RGB
for image_batch,label_batch in dataset.take(1):
    print(image_batch.shape)
    print(label_batch.numpy())
```

```
(32, 128, 128, 3)
[7 4 5 2 4 5 7 3 2 8 7 0 4 0 2 7 9 7 7 1 4 3 0 5 9 8 0 2 7 3 2 0]
```

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

```
(128, 128, 3)
```

```
 # for every run of the current cell image will shuffle,used shuffle function at the loading of the data
plt.figure(figsize=(10,10))
for image_batch,label_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[label_batch[i]])
        plt.axis("off")
```




```
## split the Data into 80% for taining
                      #10% for testing
                      #10% for validation
```

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

```
400.8
```

```
train_ds=dataset.take(400)
len(train_ds)
```

```
400
```

```
test_ds=dataset.skip(400)
len(test_ds)
```

```
101
```

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

```
50.1
```

```
val_ds=test_ds.take(50)
len(val_ds)
```

```
50
```

```
test_ds=test_ds.skip(50)
len(test_ds)
```

```
51
```

```
def get_dataset_partitions_tf(ds,train_split=0.8,val_split=0.1,test_split=0.1,shuffle=True,shuffle_size=1000):

    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
```

```
#assigning the data to the test,validation,train
train_ds,val_ds,test_ds=get_dataset_partitions_tf(dataset)
```

```
# checking the data split
```

```
len(train_ds)
```

```
400
```

```
len(val_ds)
```

```
50
```

```
len(test_ds)
```

```
    51


#elements of the dataset in memory after they are loaded from disk or any other data source.
train_ds=train_ds.cache().shuffle(1000).prefetch(buffer_size=t.data.AUTOTUNE)
val_ds=val_ds.cache().shuffle(1000).prefetch(buffer_size=t.data.AUTOTUNE)
test_ds=test_ds.cache().shuffle(1000).prefetch(buffer_size=t.data.AUTOTUNE)


# scalling reduce the computing power on the model
resize_and_rescale=t.keras.Sequential([
    layers.experimental.preprocessing.Rescaling(IMAGE_SIZE,IMAGE_SIZE),
    layers.experimental.preprocessing.Rescaling(1.0/255)
])


# Training a model using randomflip and random rotation to perform better
# if it is roatated images in a test data
data_augmentation=t.keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
    layers.experimental.preprocessing.RandomRotation(0.2)
])


#input_shape=(BATCH_SIZE,IMAGE_SIZE,IMAGE_SIZE,CHANNELS)
#n_classes =10

#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,(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)
```

```
#input_shape=(BATCH_SIZE,IMAGE_SIZE,IMAGE_SIZE,CHANNELS)
#n_classes =10

#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',input_shape=input_shape),
   # layers.MaxPooling2D((2,2)),

    #layers.Conv2D(64,kernel_size=(3,3),activation='relu',input_shape=input_shape),
    #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)


#BATCH_SIZE = 32
#IMAGE_SIZE = 64  # Adjust this as needed
#CHANNELS = 3  # Assuming RGB images
#n_classes = 10

#model = models.Sequential([
#    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_SIZE, IMAGE_SIZE, CHANNELS)),
#    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.summary()
```

```
from tensorflow.keras import layers, models

BATCH_SIZE = 32
IMAGE_SIZE = 128  # Adjust this as needed
CHANNELS = 3  # Assuming RGB images
n_classes = 10

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMAGE_SIZE, IMAGE_SIZE, CHANNELS)),
    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.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896

 max_pooling2d (MaxPooling2  (None, 63, 63, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 61, 61, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 30, 30, 64)        0
 g2D)

 flatten (Flatten)           (None, 57600)             0

 dense (Dense)               (None, 64)                3686464

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 3706506 (14.14 MB)
Trainable params: 3706506 (14.14 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.summary()
```


```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 126, 126, 32)      896

 max_pooling2d (MaxPooling2  (None, 63, 63, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 61, 61, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 30, 30, 64)        0
 g2D)

 flatten (Flatten)           (None, 57600)             0

 dense (Dense)               (None, 64)                3686464

 dense_1 (Dense)             (None, 10)                650
```

```
=================================================================
Total params: 3706506 (14.14 MB)
Trainable params: 3706506 (14.14 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# The model produces probabilities, and the loss function assumes these probabilities are already obtained through a s
model.compile(
    optimizer='adam',
    loss=t.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

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

```
Epoch 22/50
400/400 [==============================] - 6s 15ms/step - loss: 0.1217 - accuracy: 0.9729 - val_loss: 0.8147 - v
Epoch 23/50
400/400 [==============================] - 6s 15ms/step - loss: 0.0345 - accuracy: 0.9912 - val_loss: 1.0895 - v
Epoch 24/50
```

```
400/400 [==============================] - 6s 15ms/step - loss: 0.0445 - accuracy: 0.9909 - val_loss: 1.2074 - v
Epoch 46/50
400/400 [==============================] - 6s 15ms/step - loss: 0.0359 - accuracy: 0.9918 - val_loss: 1.3986 - v
Epoch 47/50
400/400 [==============================] - 6s 15ms/step - loss: 0.0866 - accuracy: 0.9856 - val_loss: 1.3510 - v
Epoch 48/50
400/400 [==============================] - 6s 16ms/step - loss: 0.1512 - accuracy: 0.9766 - val_loss: 1.4744 - v
Epoch 49/50
400/400 [==============================] - 6s 15ms/step - loss: 0.0632 - accuracy: 0.9873 - val_loss: 1.4567 - v
Epoch 50/50
```

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

```
51/51 [==============================] - 48s 7ms/step - loss: 1.0629 - accuracy: 0.9271
```

```
scores
```

```
[1.062896728515625, 0.9270833134651184]
```

```
history
```

```
<keras.src.callbacks.History at 0x7b96c9a7d510>
```

```
history.params
```

```
{'verbose': 1, 'epochs': 50, 'steps': 400}
```

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

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

```
# checking the loss of first 10 epochs
history.history['loss'][:10]
```

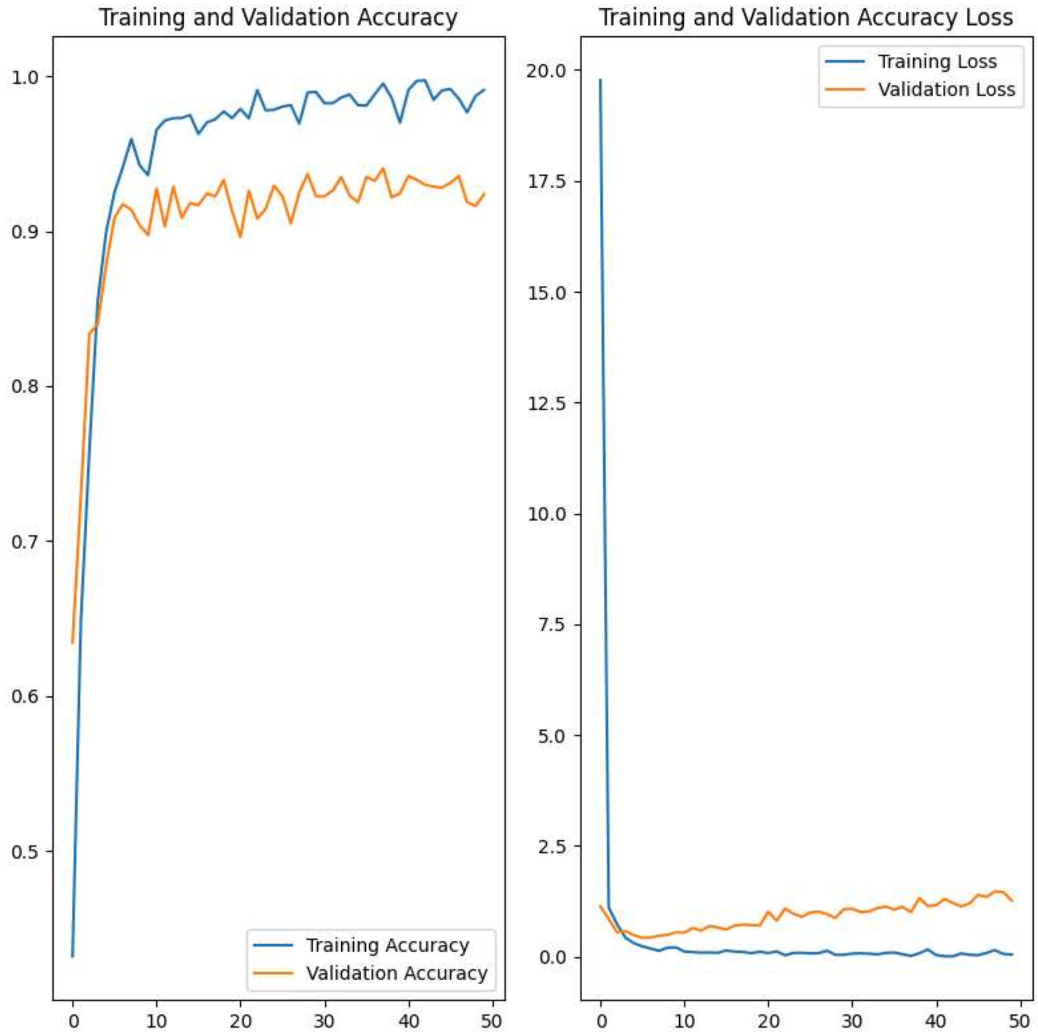
```
[19.766376495361328,
 1.1085776090621948,
 0.7291713953018188,
 0.42948082089424133,
 0.30658048391342163,
 0.23607350885868073,
 0.18448799848556519,
 0.134071484208107,
 0.20553597807884216,
 0.21177811920642853]
```

```
history.history['accuracy']
```

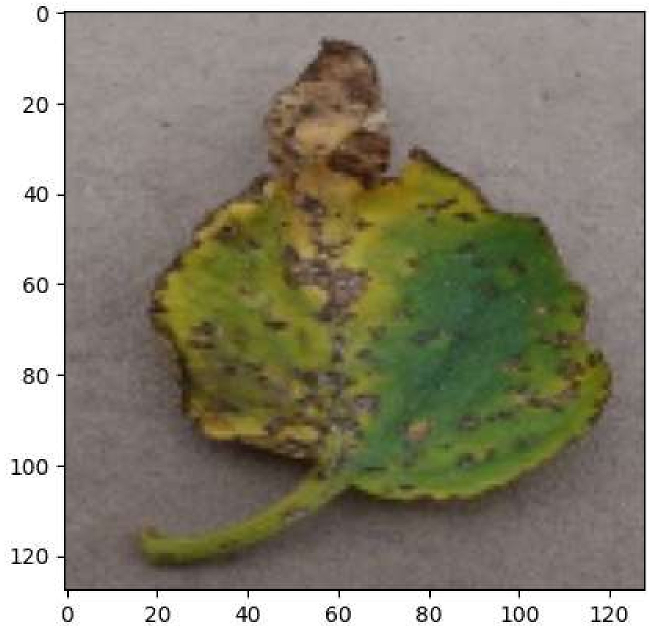
```
[0.4317968785762787,
 0.6489062309265137,
 0.7579687237739563,
 0.8548437356948853,
 0.899218738079071,
 0.9252343773841858,
 0.9414843916893005,
 0.9595312476158142,
 0.9424218535423279,
 0.9362499713897705,
 0.9654687643051147,
 0.971484363079071,
 0.9729687571525574,
 0.9731249809265137,
 0.9750000238418579,
 0.9628124833106995,
 0.9702343940734863,
 0.9722656011581421,
```

```
     0.977343738079071,
     0.9730468988418579,
     0.9789843559265137,
     0.9728906154632568,
     0.9911718964576721,
     0.9780468940734863,
     0.9784374833106995,
     0.98046875,
     0.9814062714576721,
     0.9694530963897705,
     0.9895312786102295,
     0.9899218678474426,
     0.9826562404632568,
     0.9826562404632568,
     0.986328125,
     0.98828125,
     0.9814062714576721,
     0.9810937643051147,
     0.9884374737739563,
     0.9953906536102295,
     0.9861719012260437,
     0.9700781106948853,
     0.9911718964576721,
     0.9970312714576721,
     0.9974218606948853,
     0.9848437309265137,
     0.9908593893051147,
     0.9917968511581421,
     0.9856250286102295,
     0.9766406416893005,
     0.9872656464576721,
     0.9911718964576721]
```

```
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 Accuracy Loss')
plt.tight_layout()
plt.show()
```

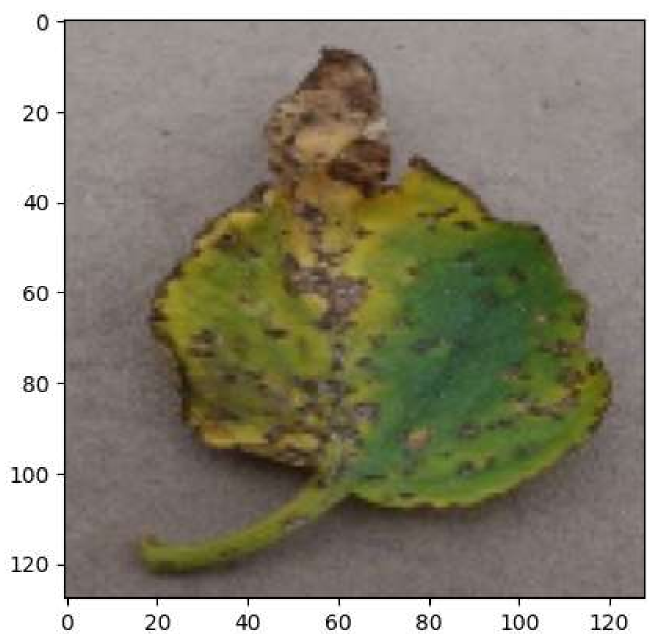



```
# Run Prediction on a sample
for images_batch,labels_batch in test_ds.take(1):

     plt.imshow(images_batch[0].numpy().astype('uint8'))
```



```
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(batch_prediction[0])
```

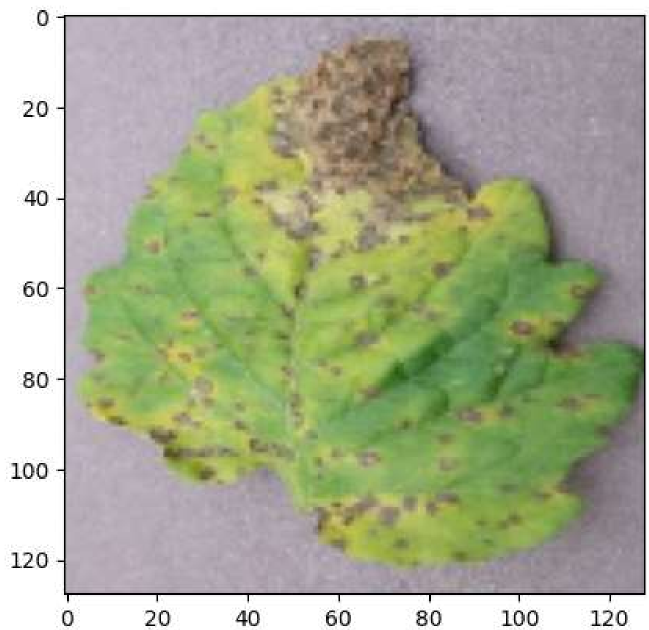
```
first image to predict
actual label: Tomato_Bacterial_spot
1/1 [==============================] - 0s 185ms/step
[9.9994338e-01 9.2219116e-20 6.9240924e-20 4.0200014e-29 4.9699920e-05
 3.5262285e-28 3.7126449e-24 6.9165922e-06 3.4596389e-33 5.5257718e-24]
```



```
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: Tomato_Septoria_leaf_spot
1/1 [==============================] - 0s 24ms/step
predicted label: Tomato_Septoria_leaf_spot
```




```
def predict(model, img):
     img_array = t.keras.preprocessing.image.img_to_array(images[i].numpy())
     img_array = t.expand_dims(img_array, 0) # Create a batch

     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
```

```
#check the multiple samples how its going to predict
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 257ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 17ms/step
```

Actual:Tomato__Target_Spot,
predicted: Tomato__Target_Spot.
Confidence: 99.96%

Actual:Tomato_Bacterial_spot,
predicted: Tomato_Bacterial_spot.
Confidence: 99.97%

Actual:Tomato__Tomato_YellowLeaf__Curl_Virus,
predicted: Tomato__Tomato_YellowLeaf__Curl_Virus.
Confidence: 100.0%

Actual:Tomato_Spider_mites_Two_spotted_spider_mite,
predicted: Tomato_Spider_mites_Two_spotted_spider_mite.
Confidence: 99.99%

Actual:Tomato_healthy,
predicted: Tomato_healthy.
Confidence: 100.0%

Actual:Tomato_healthy,
predicted: Tomato_Septoria_leaf_spot.
Confidence: 98.98%