tomato-disease-prediction

May 17, 2024

[25]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
```

0.1 Collect data and create Dataset‘

[2]:
```
IMAGE_DIRECTORY = 'PlantVillage'
```

[3]:
```
IMG_SIZE = 256
BATCH_SIZE = 32
```

[4]:
```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    IMAGE_DIRECTORY,
    color_mode='rgb',
    batch_size=BATCH_SIZE,
    image_size=(IMG_SIZE, IMG_SIZE),
    shuffle=True
)
```

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

[5]:
```
class_names = dataset.class_names
class_names # there are 10 classes
```

[5]:
```
['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']
```

[6]:
```
# check one batch
dataset.take(1)
```

[6]:
```
<_TakeDataset element_spec=(TensorSpec(shape=(None, 256, 256, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None,), dtype=tf.int32,
name=None))>
```

[7]:
```
# check the batch size, 32 images of size (256 x 256) and 3 channels
plt.figure(figsize=(15, 15))

for image_batch, label_batch in dataset.take(1):
    print(f'Image batch size: {image_batch.shape}')
    print(f'Batch labels: {label_batch.numpy()}')
    print(f'First image: shape= {image_batch[0].shape}, type=␣
 ↪{type(image_batch[0])}')

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

```
Image batch size: (32, 256, 256, 3)
Batch labels: [9 6 7 9 9 6 7 7 1 7 9 4 0 1 4 7 8 7 6 4 2 6 4 2 5 1 0 9 1 9 2 0]
First image: shape= (256, 256, 3), type= <class
'tensorflow.python.framework.ops.EagerTensor'>
```

```
2024-04-28 23:39:06.735604: W tensorflow/core/framework/local_rendezvous.cc:404]
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```

[8]:
```
def get_train_test_split(ds, train_size=0.8, val_size=0.1, test_size=0.1,␣
↪shuffle=True, shuffle_size=10000):
    dataset_size = len(ds)

    if shuffle:
        ds.shuffle(shuffle_size, seed=7)

    train_size = int(dataset_size * train_size)
    val_size = int(dataset_size * val_size)
    test_size = int(dataset_size * test_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
```

[9]:
```
train_ds, val_ds, test_ds = get_train_test_split(dataset)
print(f'Training dataset size: {len(train_ds)}')
print(f'Validation dataset size: {len(val_ds)}')
print(f'Test dataset size: {len(test_ds)}')
```

```
Training dataset size: 400
Validation dataset size: 50
Test dataset size: 51
```

[10]:
```
# if we use prefetch(), when GPU will be training a batch then CPU will fetch a␣
 ↪different batch for processing (without sitting idle)
# if we use cache(), images will be cached for processing in multiple EPOCHS

# 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)
```

0.2 Create and run the Model

[11]:
```
# Create resize and rescale layers

resize_and_rescale = tf.keras.Sequential([
    tf.keras.layers.Resizing(IMG_SIZE, IMG_SIZE),
    tf.keras.layers.Rescaling(1.0/255)
], name='resize_and_rescale')
```

[12]:
```
# Create data augmentation layers
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip("horizontal_and_vertical"),
  tf.keras.layers.RandomRotation(0.2)
], name='data_augmentation')
```

[13]:
```
CHANNELS = 3
input_shape = (BATCH_SIZE, IMG_SIZE, IMG_SIZE, CHANNELS)
num_classes = len(class_names)

model = tf.keras.Sequential([
    resize_and_rescale,
    data_augmentation,
    tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu',␣
 ↪name='Conv2D_1', input_shape=input_shape),
    tf.keras.layers.MaxPooling2D((2,2), name='MaxPool2D_1'),
    tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu',␣
 ↪name='Conv2D_2'),
    tf.keras.layers.MaxPooling2D((2,2), name='MaxPool2D_2'),
```

```
    tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu',␣
 ↪name='Conv2D_3'),
    tf.keras.layers.MaxPooling2D((2,2), name='MaxPool2D_3'),
    tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu',␣
 ↪name='Conv2D_4'),
    tf.keras.layers.MaxPooling2D((2,2), name='MaxPool2D_4'),
    tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu',␣
 ↪name='Conv2D_5'),
    tf.keras.layers.MaxPooling2D((2,2), name='MaxPool2D_5'),
    tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu',␣
 ↪name='Conv2D_6'),
    tf.keras.layers.MaxPooling2D((2,2), name='MaxPool2D_6'),
    tf.keras.layers.Flatten(name='Flatten_Layer'),
    tf.keras.layers.Dense(64, activation='relu', name='Dense_1'),
    tf.keras.layers.Dense(num_classes, activation='softmax', name='Dense_2')
])

model.build(input_shape=input_shape)
```

```
/opt/anaconda3/envs/python311cv/lib/python3.11/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[14]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
resize_and_rescale (Sequential)	(32, 256, 256, 3)	0
data_augmentation (Sequential)	(32, 256, 256, 3)	0
Conv2D_1 (Conv2D)	(32, 254, 254, 32)	896
MaxPool2D_1 (MaxPooling2D)	(32, 127, 127, 32)	0
Conv2D_2 (Conv2D)	(32, 125, 125, 64)	18,496
MaxPool2D_2 (MaxPooling2D)	(32, 62, 62, 64)	0
Conv2D_3 (Conv2D)	(32, 60, 60, 64)	36,928

```
 MaxPool2D_3 (MaxPooling2D)        (32, 30, 30, 64)                0

 Conv2D_4 (Conv2D)                 (32, 28, 28, 64)           36,928

 MaxPool2D_4 (MaxPooling2D)        (32, 14, 14, 64)                0

 Conv2D_5 (Conv2D)                 (32, 12, 12, 64)           36,928

 MaxPool2D_5 (MaxPooling2D)        (32, 6, 6, 64)                  0

 Conv2D_6 (Conv2D)                 (32, 4, 4, 64)             36,928

 MaxPool2D_6 (MaxPooling2D)        (32, 2, 2, 64)                  0

 Flatten_Layer (Flatten)           (32, 256)                       0

 Dense_1 (Dense)                   (32, 64)                   16,448

 Dense_2 (Dense)                   (32, 10)                      650


Total params: 184,202 (719.54 KB)

Trainable params: 184,202 (719.54 KB)

Non-trainable params: 0 (0.00 B)
```

[15]:
```
# compile the model

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

[16]:
```
EPOCHS = 50

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

```
Epoch 1/50
400/400             289s 722ms/step -
```

```
accuracy: 0.2646 - loss: 2.0098 - val_accuracy: 0.3975 - val_loss: 1.8658
Epoch 2/50
400/400             300s 748ms/step -
accuracy: 0.5422 - loss: 1.2684 - val_accuracy: 0.5381 - val_loss: 1.3434
Epoch 3/50
400/400             285s 713ms/step -
accuracy: 0.6904 - loss: 0.8898 - val_accuracy: 0.7700 - val_loss: 0.6717
Epoch 4/50
400/400             289s 722ms/step -
accuracy: 0.7809 - loss: 0.6288 - val_accuracy: 0.6388 - val_loss: 1.7585
Epoch 5/50
400/400             285s 712ms/step -
accuracy: 0.8180 - loss: 0.5295 - val_accuracy: 0.7619 - val_loss: 0.8448
Epoch 6/50
400/400             285s 713ms/step -
accuracy: 0.8630 - loss: 0.3960 - val_accuracy: 0.7831 - val_loss: 0.6783
Epoch 7/50
400/400             329s 824ms/step -
accuracy: 0.8920 - loss: 0.3146 - val_accuracy: 0.8531 - val_loss: 0.4184
Epoch 9/50
400/400             682s 2s/step -
accuracy: 0.8937 - loss: 0.3006 - val_accuracy: 0.7750 - val_loss: 0.8846
Epoch 10/50
400/400             1543s 4s/step -
accuracy: 0.9170 - loss: 0.2400 - val_accuracy: 0.8900 - val_loss: 0.3292
Epoch 11/50
400/400             3555s 9s/step -
accuracy: 0.9144 - loss: 0.2524 - val_accuracy: 0.8875 - val_loss: 0.3507
Epoch 12/50
400/400             293s 731ms/step -
accuracy: 0.9248 - loss: 0.2159 - val_accuracy: 0.8587 - val_loss: 0.4345
Epoch 13/50
400/400             260s 649ms/step -
accuracy: 0.9384 - loss: 0.1834 - val_accuracy: 0.9056 - val_loss: 0.3230
Epoch 15/50
400/400             2654s 7s/step -
accuracy: 0.9321 - loss: 0.1969 - val_accuracy: 0.8419 - val_loss: 0.4752
Epoch 16/50
400/400             3059s 8s/step -
accuracy: 0.9376 - loss: 0.1839 - val_accuracy: 0.9106 - val_loss: 0.2543
Epoch 17/50
400/400             3015s 8s/step -
accuracy: 0.9470 - loss: 0.1540 - val_accuracy: 0.9131 - val_loss: 0.2385
Epoch 18/50
400/400             1265s 3s/step -
accuracy: 0.9461 - loss: 0.1556 - val_accuracy: 0.9087 - val_loss: 0.3099
Epoch 19/50
400/400             3197s 8s/step -
```

```
accuracy: 0.9413 - loss: 0.1724 - val_accuracy: 0.9425 - val_loss: 0.1839
Epoch 20/50
400/400             2234s 6s/step -
accuracy: 0.9538 - loss: 0.1377 - val_accuracy: 0.8863 - val_loss: 0.3864
Epoch 21/50
400/400             2409s 6s/step -
accuracy: 0.9589 - loss: 0.1284 - val_accuracy: 0.9294 - val_loss: 0.2088
Epoch 22/50
400/400             530s 1s/step -
accuracy: 0.9465 - loss: 0.1461 - val_accuracy: 0.9306 - val_loss: 0.2330
Epoch 23/50
400/400             327s 815ms/step -
accuracy: 0.9576 - loss: 0.1250 - val_accuracy: 0.9388 - val_loss: 0.1813
Epoch 24/50
400/400             299s 746ms/step -
accuracy: 0.9483 - loss: 0.1492 - val_accuracy: 0.9262 - val_loss: 0.2343
Epoch 25/50
400/400             294s 734ms/step -
accuracy: 0.9603 - loss: 0.1145 - val_accuracy: 0.9362 - val_loss: 0.1985
Epoch 26/50
400/400             299s 747ms/step -
accuracy: 0.9689 - loss: 0.0928 - val_accuracy: 0.9513 - val_loss: 0.1637
Epoch 27/50
400/400             275s 687ms/step -
accuracy: 0.9505 - loss: 0.1337 - val_accuracy: 0.9581 - val_loss: 0.1180
Epoch 28/50
400/400             273s 683ms/step -
accuracy: 0.9606 - loss: 0.1117 - val_accuracy: 0.9506 - val_loss: 0.1451
Epoch 29/50
400/400             266s 665ms/step -
accuracy: 0.9646 - loss: 0.1077 - val_accuracy: 0.9531 - val_loss: 0.1363
Epoch 30/50
400/400             1416s 4s/step -
accuracy: 0.9659 - loss: 0.0922 - val_accuracy: 0.9513 - val_loss: 0.1566
Epoch 31/50
400/400             278s 694ms/step -
accuracy: 0.9526 - loss: 0.1393 - val_accuracy: 0.9581 - val_loss: 0.1232
Epoch 32/50
400/400             277s 692ms/step -
accuracy: 0.9645 - loss: 0.0966 - val_accuracy: 0.9488 - val_loss: 0.1701
Epoch 33/50
400/400             285s 712ms/step -
accuracy: 0.9664 - loss: 0.0977 - val_accuracy: 0.9506 - val_loss: 0.1536
Epoch 34/50
400/400             355s 887ms/step -
accuracy: 0.9628 - loss: 0.1138 - val_accuracy: 0.9737 - val_loss: 0.0829
Epoch 35/50
400/400             1101s 3s/step -
```

```
accuracy: 0.9682 - loss: 0.0873 - val_accuracy: 0.9450 - val_loss: 0.1960
Epoch 37/50
400/400             298s 745ms/step -
accuracy: 0.9650 - loss: 0.0995 - val_accuracy: 0.9413 - val_loss: 0.1786
Epoch 38/50
400/400             332s 831ms/step -
accuracy: 0.9708 - loss: 0.0847 - val_accuracy: 0.9681 - val_loss: 0.0989
Epoch 39/50
400/400             304s 759ms/step -
accuracy: 0.9725 - loss: 0.0876 - val_accuracy: 0.9787 - val_loss: 0.0701
Epoch 40/50
400/400             277s 693ms/step -
accuracy: 0.9673 - loss: 0.1020 - val_accuracy: 0.9344 - val_loss: 0.2304
Epoch 41/50
400/400             313s 782ms/step -
accuracy: 0.9658 - loss: 0.1016 - val_accuracy: 0.9369 - val_loss: 0.2040
Epoch 42/50
400/400             285s 713ms/step -
accuracy: 0.9673 - loss: 0.0880 - val_accuracy: 0.9131 - val_loss: 0.3001
Epoch 43/50
400/400             330s 825ms/step -
accuracy: 0.9708 - loss: 0.0904 - val_accuracy: 0.9613 - val_loss: 0.1391
Epoch 44/50
400/400             327s 818ms/step -
accuracy: 0.9760 - loss: 0.0737 - val_accuracy: 0.9681 - val_loss: 0.0936
Epoch 45/50
400/400             310s 773ms/step -
accuracy: 0.9728 - loss: 0.0817 - val_accuracy: 0.9688 - val_loss: 0.0866
Epoch 46/50
400/400             308s 771ms/step -
accuracy: 0.9729 - loss: 0.0857 - val_accuracy: 0.9638 - val_loss: 0.0893
Epoch 47/50
400/400             614s 2s/step -
accuracy: 0.9719 - loss: 0.0802 - val_accuracy: 0.9725 - val_loss: 0.0920
Epoch 48/50
400/400             319s 796ms/step -
accuracy: 0.9712 - loss: 0.0793 - val_accuracy: 0.9563 - val_loss: 0.1540
Epoch 49/50
400/400             426s 1s/step -
accuracy: 0.9710 - loss: 0.0857 - val_accuracy: 0.9688 - val_loss: 0.0916
Epoch 50/50
400/400             472s 1s/step -
accuracy: 0.9769 - loss: 0.0749 - val_accuracy: 0.9431 - val_loss: 0.1640
```

0.3 Evaluate the Model

[22]:
```
model.evaluate(test_ds)
```

```
51/51              12s 189ms/step -
accuracy: 0.9390 - loss: 0.2109
```

[22]:
```
[0.23103930056095123, 0.9348230957984924]
```

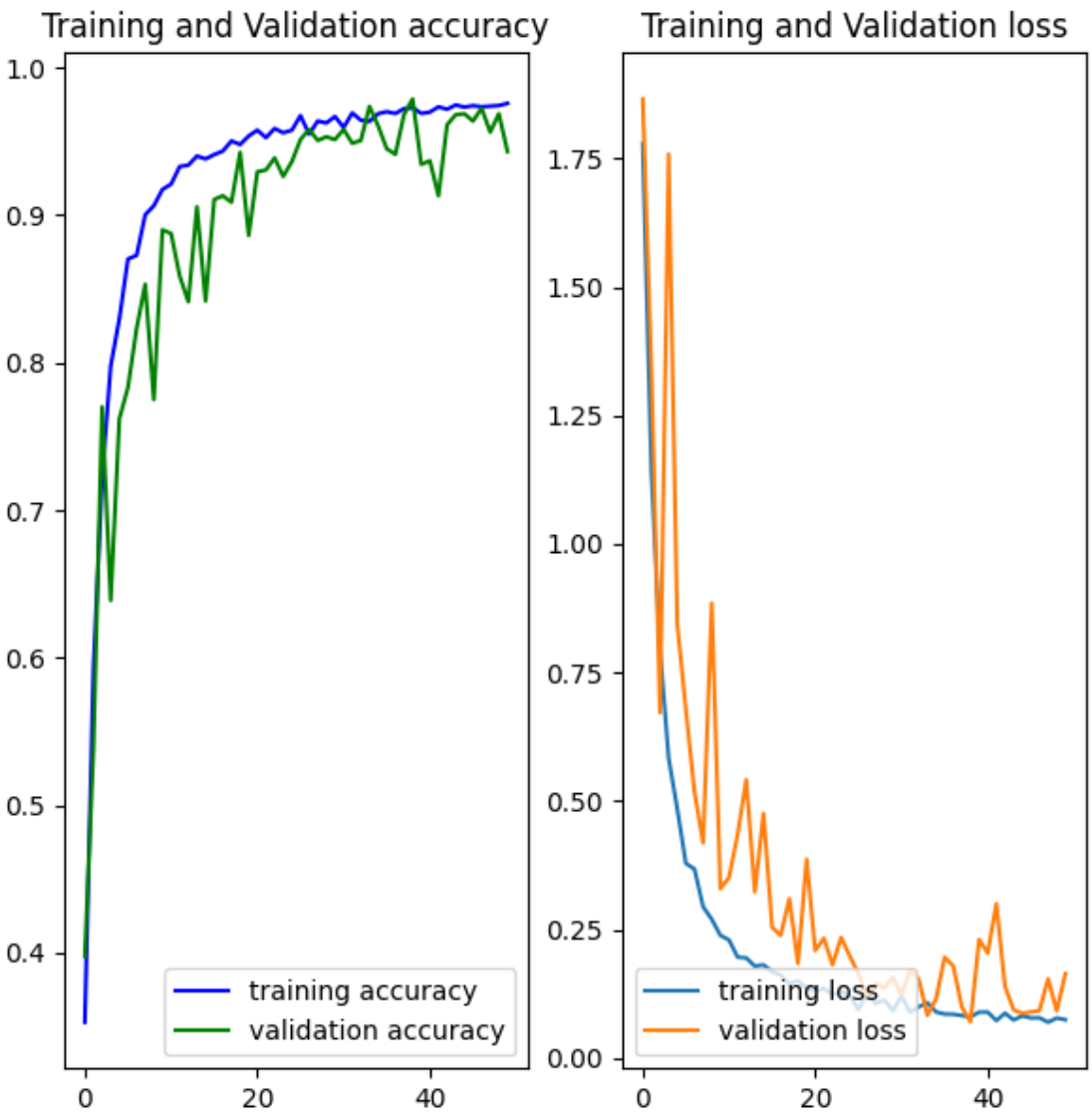
[23]:
```
training_accuracy = history.history['accuracy']
training_loss = history.history['loss']
validation_accuracy = history.history['val_accuracy']
validation_loss = history.history['val_loss']
```

[21]:
```
plt.figure(figsize=(7, 7))

plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), training_accuracy, color='blue', label='training␣
 ↪accuracy')
plt.plot(range(EPOCHS), validation_accuracy, color='green', label='validation␣
 ↪accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), training_loss, label='training loss')
plt.plot(range(EPOCHS), validation_loss, label='validation loss')
plt.legend(loc='lower left')
plt.title('Training and Validation loss')
```

[21]:
```
Text(0.5, 1.0, 'Training and Validation loss')
```



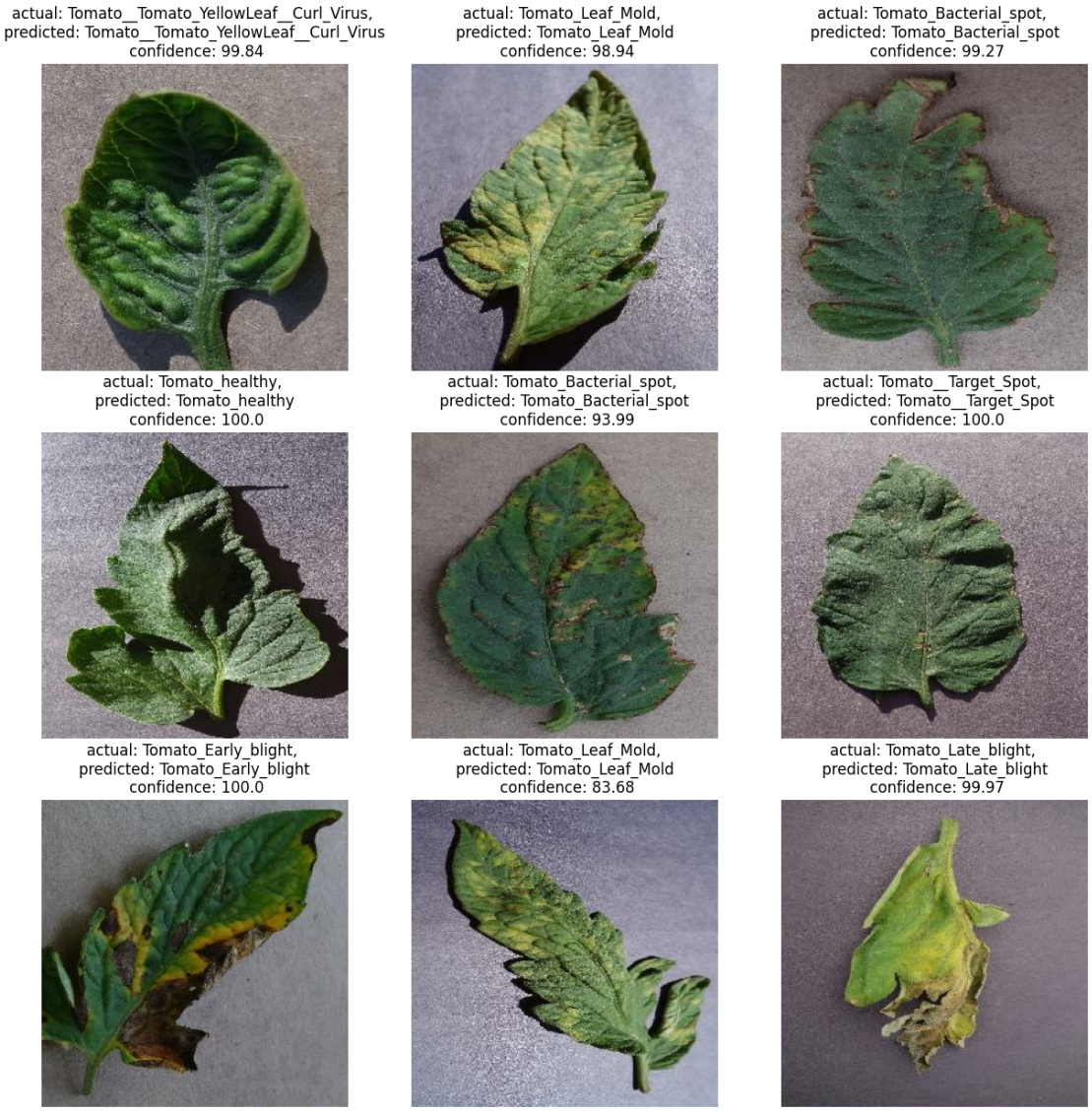
[26]:
```
# Do some testing with the testing images

plt.figure(figsize=(15, 15))
for image_batch, label_batch in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        actual_label = class_names[label_batch[i].numpy()]
        predicted_label = class_names[np.argmax(model.predict(image_batch)[i])]
        prediction_confidence =  round(np.max(model.predict(image_batch)[i]) *␣
 ↪100, 2)
```

```
        plt.title(f'actual: {actual_label}, \n predicted: {predicted_label} \n␣
↪confidence: {prediction_confidence}')
        plt.axis('off')
```

```
1/1             0s 255ms/step
1/1             0s 176ms/step
1/1             0s 177ms/step
1/1             0s 179ms/step
1/1             0s 177ms/step
1/1             0s 178ms/step
1/1             0s 172ms/step
1/1             0s 173ms/step
1/1             0s 172ms/step
1/1             0s 177ms/step
1/1             0s 178ms/step
1/1             0s 187ms/step
1/1             0s 189ms/step
1/1             0s 171ms/step
1/1             0s 174ms/step
1/1             0s 177ms/step
1/1             0s 178ms/step
1/1             0s 173ms/step
```

```
2024-04-29 10:19:09.926909: W tensorflow/core/framework/local_rendezvous.cc:404]
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```



[27]:
```
# create a method where given the model and an image will predict the label

def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0) # create a batch

    predictions = model.predict(img_array)
    predicted_label = class_names[np.argmax(model.predict(image_batch)[i])]
    prediction_confidence =  round(np.max(model.predict(image_batch)[i]) * 100,␣
 ↪2)
    return predicted_label, prediction_confidence
```

[28]:
```
predict(model, image_batch[0])
```

```
1/1             0s 50ms/step
1/1             0s 185ms/step
1/1             0s 173ms/step
```

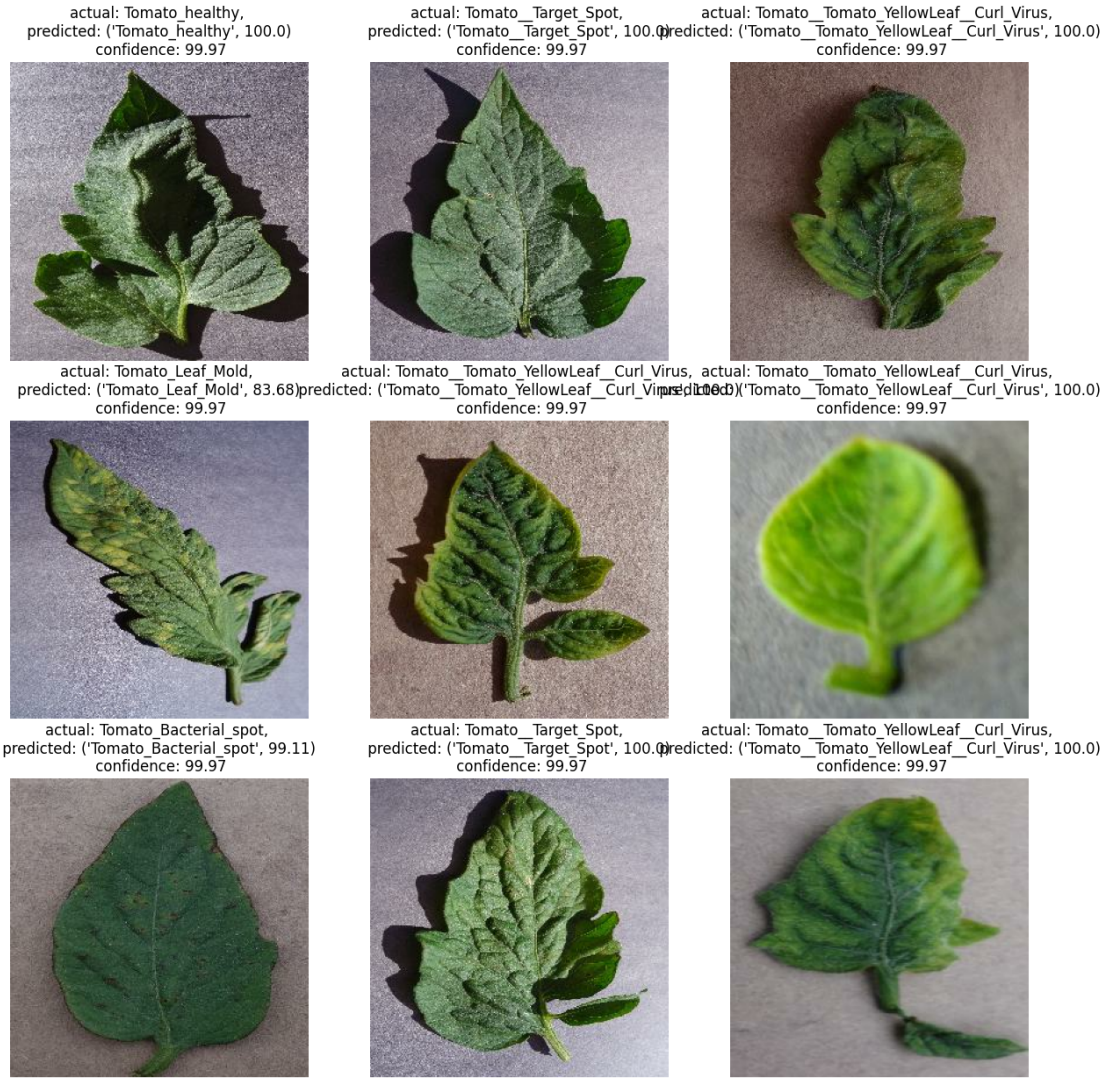
[28]:
```
('Tomato_Late_blight', 99.97)
```

[29]:
```
plt.figure(figsize=(15, 15))
for image_batch, label_batch in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i+1)
        plt.imshow(image_batch[i].numpy().astype('uint8'))
        actual_label = class_names[label_batch[i].numpy()]
        predicted_label = predict(model, image_batch[i])
        plt.title(f'actual: {actual_label}, \n predicted: {predicted_label} \n␣
 ↪confidence: {prediction_confidence}')
        plt.axis('off')
```

```
1/1             0s 16ms/step
1/1             0s 222ms/step
1/1             0s 214ms/step
1/1             0s 19ms/step
1/1             0s 198ms/step
1/1             0s 197ms/step
1/1             0s 18ms/step
1/1             0s 192ms/step
1/1             0s 198ms/step
1/1             0s 18ms/step
1/1             0s 200ms/step
1/1             0s 200ms/step
1/1             0s 18ms/step
1/1             0s 197ms/step
1/1             0s 203ms/step
1/1             0s 19ms/step
1/1             0s 213ms/step
1/1             0s 224ms/step
1/1             0s 20ms/step
1/1             0s 216ms/step
1/1             0s 219ms/step
1/1             0s 19ms/step
1/1             0s 214ms/step
1/1             0s 221ms/step
1/1             0s 20ms/step
1/1             0s 214ms/step
1/1             0s 217ms/step

2024-04-29 10:19:41.737325: W tensorflow/core/framework/local_rendezvous.cc:404]
```

```
Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
```



[30]:
```
!mkdir saved_models
```

0.4 Save the Model

[33]:
```
# check the latest model version existing

import os

model_version = max([int(os.path.splitext(x)[0]) for x in os.
  ↪listdir('saved_models') + [str(0)]]) + 1
model_version
```

[33]: 1

[34]:
```
model_name = f'saved_models/{model_version}.keras'
print(model_name)
model.save(model_name)
```

```
saved_models/1.keras
```

[ ]: