```
pwd
```

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

```
!ls "/content/drive/My Drive/Plant Diseases Detector"
```

```
from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf
#tf.logging.set_verbosity(tf.logging.ERROR)
tf.enable_eager_execution()

import tensorflow_hub as hub
import os
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers
#from keras import optimizers
```

```
# verify TensorFlow version

print("Version: ", tf.__version__)
print("Eager mode: ", tf.executing_eagerly())
print("Hub version: ", hub.__version__)
print("GPU is", "available" if tf.test.is_gpu_available() else "NOT AVAILABLE")
```

Load the data

We will download a public dataset of 54,305 images of diseased and healthy plant leaves collected under controlled conditions (PlantVillage Dataset). The images cover 14 species of crops, including: apple, blueberry, cherry, grape, orange, peach, pepper, potato, raspberry, soy, squash, strawberry and tomato. It contains images of 17 basic diseases, 4 bacterial diseases, 2 diseases caused by mold (oomycete), 2 viral diseases and 1 disease caused by a mite. 12 crop species also have healthy leaf images that are not visibly affected by disease. Then store the downloaded zip file to the "/tmp/" directory.

we'll need to make sure the input data is resized to 224x224 or 229x229 pixels as required by the networks.

```
zip_file = tf.keras.utils.get_file(origin='https://storage.googleapis.com/plantdata/PlantVillage.zip',
                                   fname='PlantVillage.zip', extract=True)
```

```
zip_file
```

Prepare training and validation dataset

Create the training and validation directories

```
data_dir = os.path.join(os.path.dirname(zip_file), 'PlantVillage')
train_dir = os.path.join(data_dir, 'train')
validation_dir = os.path.join(data_dir, 'validation')
```

```
print(train_dir)
```

```
import time
import os
from os.path import exists

def count(dir, counter=0):
    "returns number of files in dir and subdirs"
    for pack in os.walk(dir):
        for f in pack[2]:
            counter += 1
    return dir + " : " + str(counter) + "files"
```

```
print('total images for training :', count(train_dir))
print('total images for validation :', count(validation_dir))
```

Label mapping

You'll also need to load in a mapping from category label to category name. You can find this in the file `categories.json`. It's a JSON object which you can read in with the `json module`. This will give you a dictionary mapping the integer encoded categories to the actual names of the plants and diseases.

```
import json

with open('/content/drive/My Drive/Plant Diseases Detector/categories.json', 'r') as f:
    cat_to_name = json.load(f)
    classes = list(cat_to_name.values())

print (classes)
```

```
print('Number of classes:',len(classes))
```

Select the Hub/TF2 module to use

```
module_selection = ("mobilenet_v2", 224, 1280) #@param ["(\"mobilenet_v2\", 224, 1280)", "(\"inception_v3\", 299, 2048)"] {type:"ra
handle_base, pixels, FV_SIZE = module_selection
MODULE_HANDLE ="https://tfhub.dev/google/tf2-preview/{}/feature_vector/2".format(handle_base)
IMAGE_SIZE = (pixels, pixels)
print("Using {} with input size {} and output dimension {}".format(
  MODULE_HANDLE, IMAGE_SIZE, FV_SIZE))

BATCH_SIZE = 64 #@param {type:"integer"}
```

module_selection: ("mobilenet_v2", 224, 1280)

BATCH_SIZE: 64

Data Preprocessing

Let's set up data generators that will read pictures in our source folders, convert them to `float32` tensors, and feed them (with their labels) to our network.

As you may already know, data that goes into neural networks should usually be normalized in some way to make it more amenable to processing by the network. (It is uncommon to feed raw pixels into a convnet.) In our case, we will preprocess our images by normalizing the pixel values to be in the `[0, 1]` range (originally all values are in the `[0, 255]` range).

```
# Inputs are suitably resized for the selected module. Dataset augmentation (i.e., random distortions of an image each time it is r

validation_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
validation_generator = validation_datagen.flow_from_directory(
    validation_dir,
    shuffle=False,
    seed=42,
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE)

do_data_augmentation = True #@param {type:"boolean"}
if do_data_augmentation:
  train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
      rescale = 1./255,
      rotation_range=40,
      horizontal_flip=True,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      fill_mode='nearest' )
else:
  train_datagen = validation_datagen

train_generator = train_datagen.flow_from_directory(
    train_dir,
    subset="training",
    shuffle=True,
    seed=42,
    color_mode="rgb",
    class_mode="categorical",
    target_size=IMAGE_SIZE,
    batch_size=BATCH_SIZE)

```

do_data_augmentation: ☑

```
Found 10861 images belonging to 38 classes.
Found 43444 images belonging to 38 classes.
```

Build the model

All it takes is to put a linear classifier on top of the feature_extractor_layer with the Hub module.

For speed, we start out with a non-trainable feature_extractor_layer, but you can also enable fine-tuning for greater accuracy.

```
feature_extractor = hub.KerasLayer(MODULE_HANDLE,
                                   input_shape=IMAGE_SIZE+(3,),
                                   output_shape=[FV_SIZE])
```

```
do_fine_tuning = False #@param {type:"boolean"}
base_model = module_selection
if do_fine_tuning:
    feature_extractor.trainable = True
    # unfreeze some layers of base network for fine-tuning
    for layer in base_model.layers[-30:]:
        layer.trainable =True

else:
    feature_extractor.trainable = False
```

do_fine_tuning: ■

```
print("Building model with", MODULE_HANDLE)
model = tf.keras.Sequential([
    feature_extractor,
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(rate=0.2),
    tf.keras.layers.Dense(train_generator.num_classes, activation='softmax',
                          kernel_regularizer=tf.keras.regularizers.l2(0.0001))
])
#model.build((None,)+IMAGE_SIZE+(3,))

model.summary()
```

```
Building model with https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/2
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984   
_________________________________________________________________
flatten (Flatten)            (None, 1280)              0         
_________________________________________________________________
dense (Dense)                (None, 512)               655872    
_________________________________________________________________
dropout (Dropout)            (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 38)                19494     
=================================================================
Total params: 2,933,350
Trainable params: 675,366
Non-trainable params: 2,257,984
_________________________________________________________________
```

Specify Loss Function and Optimizer

```
#Compile model specifying the optimizer learning rate

LEARNING_RATE = 0.001 #@param {type:"number"}

model.compile(
   optimizer=tf.keras.optimizers.Adam(lr=LEARNING_RATE),
   loss='categorical_crossentropy',
   metrics=['accuracy'])
```

LEARNING_RATE: 0.001

Train Model

train model using validation dataset for validate each steps

```
EPOCHS=30 #@param {type:"integer"}

history = model.fit_generator(
        train_generator,
        steps_per_epoch=train_generator.samples//train_generator.batch_size,
        epochs=EPOCHS,
        validation_data=validation_generator,
        validation_steps=validation_generator.samples//validation_generator.batch_size)
```

EPOCHS: 30

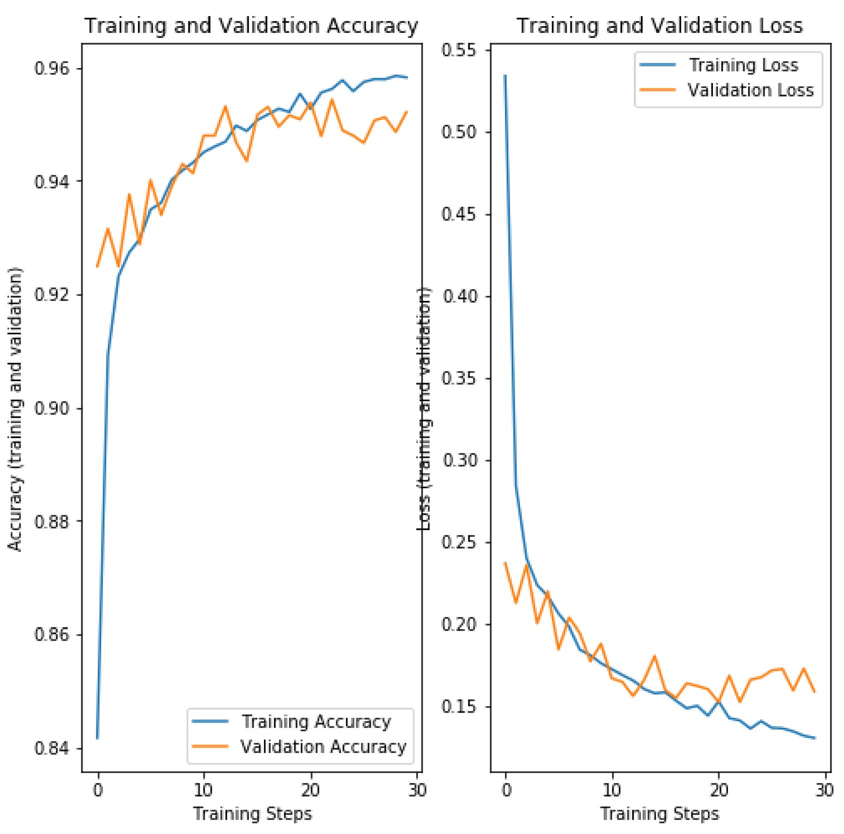
```
Epoch 1/30
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/math_grad.py:1394: where (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/math_grad.py:1394: where (from tensorflow.python.ops.array_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
677/678 [============================>.] - ETA: 0s - loss: 0.5344 - acc: 0.8416Epoch 1/30
678/678 [==============================] - 594s 876ms/step - loss: 0.5340 - acc: 0.8417 - val_loss: 0.2367 - val_acc: 0.9249
Epoch 2/30
677/678 [============================>.] - ETA: 0s - loss: 0.2840 - acc: 0.9092Epoch 1/30
678/678 [==============================] - 589s 869ms/step - loss: 0.2842 - acc: 0.9092 - val_loss: 0.2125 - val_acc: 0.9316
Epoch 3/30
677/678 [============================>.] - ETA: 0s - loss: 0.2399 - acc: 0.9232Epoch 1/30
678/678 [==============================] - 586s 865ms/step - loss: 0.2399 - acc: 0.9232 - val_loss: 0.2357 - val_acc: 0.9249
Epoch 4/30
677/678 [============================>.] - ETA: 0s - loss: 0.2235 - acc: 0.9273Epoch 1/30
678/678 [==============================] - 588s 867ms/step - loss: 0.2234 - acc: 0.9274 - val_loss: 0.2001 - val_acc: 0.9376
Epoch 5/30
677/678 [============================>.] - ETA: 0s - loss: 0.2169 - acc: 0.9297Epoch 1/30
678/678 [==============================] - 585s 862ms/step - loss: 0.2167 - acc: 0.9298 - val_loss: 0.2196 - val_acc: 0.9288
Epoch 6/30
677/678 [============================>.] - ETA: 0s - loss: 0.2063 - acc: 0.9348Epoch 1/30
678/678 [==============================] - 586s 865ms/step - loss: 0.2062 - acc: 0.9349 - val_loss: 0.1844 - val_acc: 0.9401
Epoch 7/30
677/678 [============================>.] - ETA: 0s - loss: 0.1981 - acc: 0.9361Epoch 1/30
678/678 [==============================] - 585s 863ms/step - loss: 0.1981 - acc: 0.9361 - val_loss: 0.2036 - val_acc: 0.9340
Epoch 8/30
677/678 [============================>.] - ETA: 0s - loss: 0.1843 - acc: 0.9402Epoch 1/30
678/678 [==============================] - 586s 865ms/step - loss: 0.1843 - acc: 0.9402 - val_loss: 0.1944 - val_acc: 0.9390
Epoch 9/30
677/678 [============================>.] - ETA: 0s - loss: 0.1811 - acc: 0.9417Epoch 1/30
678/678 [==============================] - 583s 860ms/step - loss: 0.1808 - acc: 0.9418 - val_loss: 0.1771 - val_acc: 0.9430
Epoch 10/30
677/678 [============================>.] - ETA: 0s - loss: 0.1759 - acc: 0.9432Epoch 1/30
678/678 [==============================] - 587s 865ms/step - loss: 0.1760 - acc: 0.9432 - val_loss: 0.1879 - val_acc: 0.9413
Epoch 11/30
677/678 [============================>.] - ETA: 0s - loss: 0.1725 - acc: 0.9450Epoch 1/30
678/678 [==============================] - 586s 864ms/step - loss: 0.1724 - acc: 0.9450 - val_loss: 0.1668 - val_acc: 0.9479
Epoch 12/30
677/678 [============================>.] - ETA: 0s - loss: 0.1688 - acc: 0.9461Epoch 1/30
678/678 [==============================] - 590s 870ms/step - loss: 0.1689 - acc: 0.9460 - val_loss: 0.1646 - val_acc: 0.9479
Epoch 13/30
677/678 [============================>.] - ETA: 0s - loss: 0.1657 - acc: 0.9468Epoch 1/30
678/678 [==============================] - 589s 869ms/step - loss: 0.1655 - acc: 0.9469 - val_loss: 0.1560 - val_acc: 0.9531
Epoch 14/30
677/678 [============================>.] - ETA: 0s - loss: 0.1603 - acc: 0.9497Epoch 1/30
678/678 [==============================] - 587s 866ms/step - loss: 0.1603 - acc: 0.9497 - val_loss: 0.1657 - val_acc: 0.9467
Epoch 15/30
677/678 [============================>.] - ETA: 0s - loss: 0.1576 - acc: 0.9487Epoch 1/30
678/678 [==============================] - 585s 863ms/step - loss: 0.1576 - acc: 0.9488 - val_loss: 0.1804 - val_acc: 0.9434
Epoch 16/30
677/678 [============================>.] - ETA: 0s - loss: 0.1582 - acc: 0.9507Epoch 1/30
678/678 [==============================] - 586s 864ms/step - loss: 0.1582 - acc: 0.9507 - val_loss: 0.1595 - val_acc: 0.9516
Epoch 17/30
677/678 [============================>.] - ETA: 0s - loss: 0.1531 - acc: 0.9517Epoch 1/30
678/678 [==============================] - 594s 875ms/step - loss: 0.1531 - acc: 0.9517 - val_loss: 0.1548 - val_acc: 0.9530
Epoch 18/30
677/678 [============================>.] - ETA: 0s - loss: 0.1484 - acc: 0.9527Epoch 1/30
678/678 [==============================] - 603s 889ms/step - loss: 0.1484 - acc: 0.9527 - val_loss: 0.1637 - val_acc: 0.9495
Epoch 19/30
677/678 [============================>.] - ETA: 0s - loss: 0.1499 - acc: 0.9521Epoch 1/30
678/678 [==============================] - 593s 875ms/step - loss: 0.1500 - acc: 0.9521 - val_loss: 0.1621 - val_acc: 0.9516
Epoch 20/30
677/678 [============================>.] - ETA: 0s - loss: 0.1438 - acc: 0.9554Epoch 1/30
678/678 [==============================] - 588s 867ms/step - loss: 0.1439 - acc: 0.9553 - val_loss: 0.1602 - val_acc: 0.9508
Epoch 21/30
677/678 [============================>.] - ETA: 0s - loss: 0.1527 - acc: 0.9526Epoch 1/30
678/678 [==============================] - 588s 867ms/step - loss: 0.1527 - acc: 0.9526 - val_loss: 0.1523 - val_acc: 0.9538
Epoch 22/30
677/678 [============================>.] - ETA: 0s - loss: 0.1426 - acc: 0.9556Epoch 1/30
678/678 [==============================] - 594s 876ms/step - loss: 0.1425 - acc: 0.9556 - val_loss: 0.1684 - val_acc: 0.9479
Epoch 23/30
677/678 [============================>.] - ETA: 0s - loss: 0.1411 - acc: 0.9562Epoch 1/30
678/678 [==============================] - 598s 883ms/step - loss: 0.1411 - acc: 0.9562 - val_loss: 0.1522 - val_acc: 0.9543
Epoch 24/30
677/678 [============================>.] - ETA: 0s - loss: 0.1361 - acc: 0.9578Epoch 1/30
678/678 [==============================] - 589s 869ms/step - loss: 0.1361 - acc: 0.9577 - val_loss: 0.1660 - val_acc: 0.9489
Epoch 25/30
677/678 [============================>.] - ETA: 0s - loss: 0.1407 - acc: 0.9558Epoch 1/30
678/678 [==============================] - 603s 889ms/step - loss: 0.1407 - acc: 0.9558 - val_loss: 0.1674 - val_acc: 0.9479
Epoch 26/30
677/678 [============================>.] - ETA: 0s - loss: 0.1366 - acc: 0.9574Epoch 1/30
678/678 [==============================] - 599s 883ms/step - loss: 0.1365 - acc: 0.9574 - val_loss: 0.1716 - val_acc: 0.9467
Epoch 27/30
677/678 [============================>.] - ETA: 0s - loss: 0.1362 - acc: 0.9579Epoch 1/30
678/678 [==============================] - 597s 880ms/step - loss: 0.1362 - acc: 0.9579 - val_loss: 0.1725 - val_acc: 0.9506
Epoch 28/30
677/678 [============================>.] - ETA: 0s - loss: 0.1344 - acc: 0.9579Epoch 1/30
678/678 [==============================] - 588s 868ms/step - loss: 0.1344 - acc: 0.9579 - val_loss: 0.1593 - val_acc: 0.9512
Epoch 29/30
677/678 [============================>.] - ETA: 0s - loss: 0.1315 - acc: 0.9586Epoch 1/30
678/678 [==============================] - 604s 891ms/step - loss: 0.1317 - acc: 0.9585 - val_loss: 0.1728 - val_acc: 0.9486
Epoch 30/30
677/678 [============================>.] - ETA: 0s - loss: 0.1302 - acc: 0.9583Epoch 1/30
678/678 [==============================] - 608s 897ms/step - loss: 0.1304 - acc: 0.9582 - val_loss: 0.1588 - val_acc: 0.9521
```

Check Performance

Plot training and validation accuracy and loss

```
import matplotlib.pylab as plt
import numpy as np

```

```python
acc = history.history['acc']
val_acc = history.history['val_acc']

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

epochs_range = range(EPOCHS)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.ylabel("Accuracy (training and validation)")
plt.xlabel("Training Steps")

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.ylabel("Loss (training and validation)")
plt.xlabel("Training Steps")
plt.show()
```



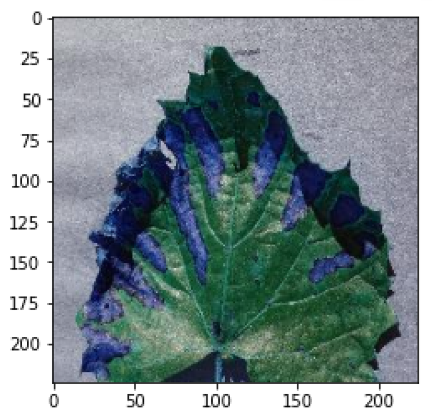
Random test

Random sample images from validation dataset and predict

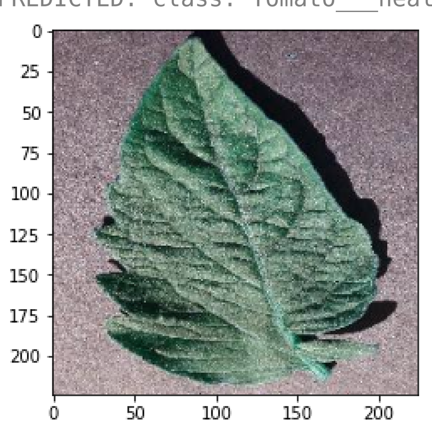
```python
# Import OpenCV
import cv2

# Utility
import itertools
import random
from collections import Counter
from glob import iglob


def load_image(filename):
    img = cv2.imread(os.path.join(data_dir, validation_dir, filename))
    img = cv2.resize(img, (IMAGE_SIZE[0], IMAGE_SIZE[1]) )
    img = img /255

    return img


def predict(image):
    probabilities = model.predict(np.asarray([img]))[0]
    class_idx = np.argmax(probabilities)

    return {classes[class_idx]: probabilities[class_idx]}
```

```python
for idx, filename in enumerate(random.sample(validation_generator.filenames, 5)):
    print("SOURCE: class: %s, file: %s" % (os.path.split(filename)[0], filename))

    img = load_image(filename)
    prediction = predict(img)
    print("PREDICTED: class: %s, confidence: %f" % (list(prediction.keys())[0], list(prediction.values())[0]))
    plt.imshow(img)
    plt.figure(idx)
    plt.show()
```

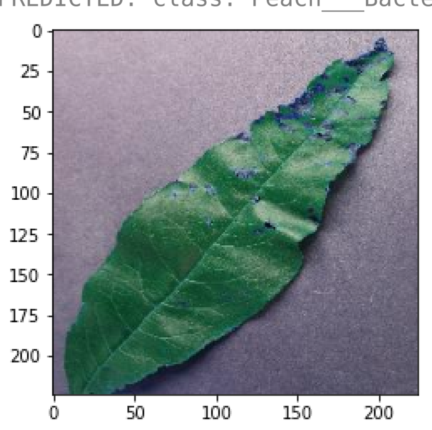
```
SOURCE: class: Grape___Esca_(Black_Measles), file: Grape___Esca_(Black_Measles)/8941a904-fa8c-4764-97d8-2a44d786dbcb___FAM_B.Msls 1331.JPG
PREDICTED: class: Grape___Esca_(Black_Measles), confidence: 0.999912
```



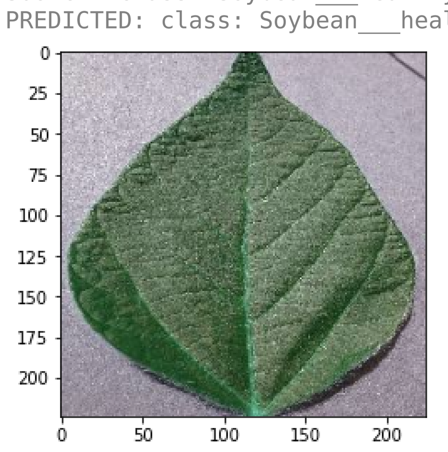
```
<Figure size 432x288 with 0 Axes>
SOURCE: class: Tomato___healthy, file: Tomato___healthy/c7f91fa4-3769-45ef-a494-1669ee62ee7a___RS_HL 9812.JPG
PREDICTED: class: Tomato___healthy, confidence: 0.977538
```



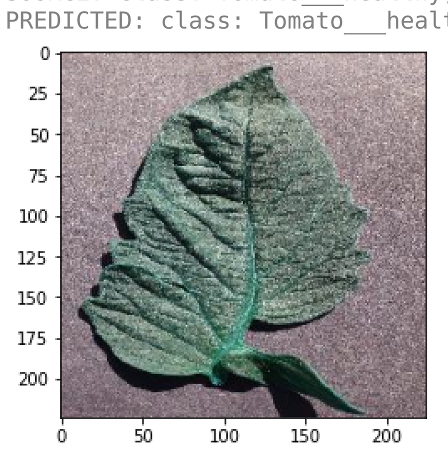
```
SOURCE: class: Peach___Bacterial_spot, file: Peach___Bacterial_spot/28eb2a2d-724b-436e-9937-95b97f74f6ac___Rut._Bact.S 1351.JPG
PREDICTED: class: Peach___Bacterial_spot, confidence: 0.999976
```



```
<Figure size 432x288 with 0 Axes>
SOURCE: class: Soybean___healthy, file: Soybean___healthy/8f6f69b5-160f-411f-bea3-bad263db0e68___RS_HL 6353.JPG
PREDICTED: class: Soybean___healthy, confidence: 1.000000
```



```
<Figure size 432x288 with 0 Axes>
SOURCE: class: Tomato___healthy, file: Tomato___healthy/5630c5b1-a956-4a34-879a-eb1cb89d22d1___RS_HL 9806.JPG
PREDICTED: class: Tomato___healthy, confidence: 0.898288
```



```
<Figure size 432x288 with 0 Axes>
```

Export as saved model and convert to TFLite

Now that you've trained the model, export it as a saved model

```python
import time
t = time.time()

export_path = "/content/drive/My Drive/Plant Diseases Detector/Model/{}".format(int(t))
tf.keras.experimental.export_saved_model(model, export_path)

export_path
```

```python
# Now confirm that we can reload it, and it still gives the same results
reloaded = tf.keras.experimental.load_from_saved_model(export_path, custom_objects={'KerasLayer':hub.KerasLayer})
```

```python
def predict_reload(image):
    probabilities = reloaded.predict(np.asarray([img]))[0]
    class_idx = np.argmax(probabilities)

    return {classes[class_idx]: probabilities[class_idx]}
```

```python
for idx, filename in enumerate(random.sample(validation_generator.filenames, 2)):
    print("SOURCE: class: %s, file: %s" % (os.path.split(filename)[0], filename))

    img = load_image(filename)
    prediction = predict_reload(img)
    print("PREDICTED: class: %s, confidence: %f" % (list(prediction.keys())[0], list(prediction.values())[0]))
    plt.imshow(img)
    plt.figure(idx)
    plt.show()
```

```python
!mkdir "/content/drive/My Drive/Plant Diseases Detector/tflite_models3"
```

```python
# convert the model to TFLite

TFLITE_MODEL = "/content/drive/My Drive/Plant Diseases Detector/tflite_models3/plant_disease_model.tflite"


# Get the concrete function from the Keras model.
run_model = tf.function(lambda x : reloaded(x))

# Save the concrete function.
concrete_func = run_model.get_concrete_function(
    tf.TensorSpec(model.inputs[0].shape, model.inputs[0].dtype)
)

# Convert the model to standard TensorFlow Lite model
converter = tf.compat.v2.lite.TFLiteConverter.from_concrete_functions([concrete_func])


converted_tflite_model = converter.convert()
open(TFLITE_MODEL, "wb").write(converted_tflite_model)
```

CONCLUSION

The model can be improved if you change some hyperparameters. You can try using a different pretrained model. It's up to you. Let me know if you can improve the accuracy! Let's develop an Android app that uses this model.