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

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

```
!unzip /content/drive/MyDrive/Tomato
```

```
Streaming output truncated to the last 5000 lines.
  inflating: Tomato/Tomato_healthy/image (342).JPG
  inflating: Tomato/Tomato_healthy/image (343).JPG
  inflating: Tomato/Tomato_healthy/image (344).JPG
  inflating: Tomato/Tomato_healthy/image (345).JPG
  inflating: Tomato/Tomato_healthy/image (346).JPG
  inflating: Tomato/Tomato_healthy/image (347).JPG
  inflating: Tomato/Tomato_healthy/image (348).JPG
  inflating: Tomato/Tomato_healthy/image (349).JPG
  inflating: Tomato/Tomato_healthy/image (35).JPG
  inflating: Tomato/Tomato_healthy/image (350).JPG
  inflating: Tomato/Tomato_healthy/image (351).JPG
  inflating: Tomato/Tomato_healthy/image (352).JPG
  inflating: Tomato/Tomato_healthy/image (353).JPG
  inflating: Tomato/Tomato_healthy/image (354).JPG
  inflating: Tomato/Tomato_healthy/image (355).JPG
  inflating: Tomato/Tomato_healthy/image (356).JPG
  inflating: Tomato/Tomato_healthy/image (357).JPG
  inflating: Tomato/Tomato_healthy/image (358).JPG
  inflating: Tomato/Tomato_healthy/image (359).JPG
  inflating: Tomato/Tomato_healthy/image (36).JPG
  inflating: Tomato/Tomato_healthy/image (360).JPG
  inflating: Tomato/Tomato_healthy/image (361).JPG
  inflating: Tomato/Tomato_healthy/image (362).JPG
  inflating: Tomato/Tomato_healthy/image (363).JPG
  inflating: Tomato/Tomato_healthy/image (364).JPG
  inflating: Tomato/Tomato_healthy/image (365).JPG
  inflating: Tomato/Tomato_healthy/image (366).JPG
  inflating: Tomato/Tomato_healthy/image (367).JPG
  inflating: Tomato/Tomato_healthy/image (368).JPG
  inflating: Tomato/Tomato_healthy/image (369).JPG
  inflating: Tomato/Tomato_healthy/image (37).JPG
  inflating: Tomato/Tomato_healthy/image (370).JPG
  inflating: Tomato/Tomato_healthy/image (371).JPG
  inflating: Tomato/Tomato_healthy/image (372).JPG
  inflating: Tomato/Tomato_healthy/image (373).JPG
  inflating: Tomato/Tomato_healthy/image (374).JPG
  inflating: Tomato/Tomato_healthy/image (375).JPG
  inflating: Tomato/Tomato_healthy/image (376).JPG
  inflating: Tomato/Tomato_healthy/image (377).JPG
  inflating: Tomato/Tomato_healthy/image (378).JPG
  inflating: Tomato/Tomato_healthy/image (379).JPG
  inflating: Tomato/Tomato_healthy/image (38).JPG
  inflating: Tomato/Tomato_healthy/image (380).JPG
  inflating: Tomato/Tomato_healthy/image (381).JPG
  inflating: Tomato/Tomato_healthy/image (382).JPG
  inflating: Tomato/Tomato_healthy/image (383).JPG
```

```
inflating: Tomato/Tomato_healthy/image (384).JPG
inflating: Tomato/Tomato_healthy/image (385).JPG
inflating: Tomato/Tomato_healthy/image (386).JPG
inflating: Tomato/Tomato_healthy/image (387).JPG
inflating: Tomato/Tomato_healthy/image (388).JPG
inflating: Tomato/Tomato_healthy/image (389).JPG
inflating: Tomato/Tomato_healthy/image (39).JPG
inflating: Tomato/Tomato_healthy/image (390).JPG
inflating: Tomato/Tomato_healthy/image (391).JPG
inflating: Tomato/Tomato_healthy/image (392).JPG
inflating: Tomato/Tomato_healthy/image (393).JPG
inflating: Tomato/Tomato_healthy/image (394).JPG
```

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing import image
import numpy as np
import os
import random
from shutil import copyfile
import matplotlib.image  as mpimg
import matplotlib.pyplot as plt
import cv2
from google.colab.patches import cv2_imshow
```

```
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Conv2D, Activation, MaxPooling2D, Dropout, Dense
from tensorflow.keras.layers import GlobalAveragePooling2D, Flatten, BatchNormalizatio
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStoppi
from keras.utils.np_utils import to_categorical
```

```
to_create = [
'leaf_disease',
'leaf_disease/training',
'leaf_disease/testing',
'leaf_disease/training/Tomato_Bacterial_spot',
'leaf_disease/training/Tomato_Late_blight',
'leaf_disease/training/Tomato_Early_blight',
'leaf_disease/training/Tomato_healthy',
'leaf_disease/training/Tomato_Yellow_Leaf_Curl_Virus',
'leaf_disease/training/Tomato_Leaf_Mold',
'leaf_disease/testing/Tomato_Bacterial_spot',
'leaf_disease/testing/Tomato_Late_blight',
'leaf_disease/testing/Tomato_Early_blight',
'leaf_disease/testing/Tomato_healthy',
'leaf_disease/testing/Tomato_Yellow_Leaf_Curl_Virus',
'leaf_disease/testing/Tomato_Leaf_Mold'
]
for directory in to_create:
```

```
  for directory in os_create:
    try:
      os.mkdir(directory)
      print(directory, 'created')
    except:
      print(directory, 'failed')
```

```
leaf_disease created
leaf_disease/training created
leaf_disease/testing created
leaf_disease/training/Tomato_Bacterial_spot created
leaf_disease/training/Tomato_Late_blight created
leaf_disease/training/Tomato_Early_blight created
leaf_disease/training/Tomato_healthy created
leaf_disease/training/Tomato_Yellow_Leaf_Curl_Virus created
leaf_disease/training/Tomato_Leaf_Mold created
leaf_disease/testing/Tomato_Bacterial_spot created
leaf_disease/testing/Tomato_Late_blight created
leaf_disease/testing/Tomato_Early_blight created
leaf_disease/testing/Tomato_healthy created
leaf_disease/testing/Tomato_Yellow_Leaf_Curl_Virus created
leaf_disease/testing/Tomato_Leaf_Mold created
```

```
def split_data(SOURCE, TRAINING, TESTING, SPLIT_SIZE):
    all_files = []

    for file_name in os.listdir(SOURCE):
        file_path = SOURCE + file_name

        if os.path.getsize(file_path):
            all_files.append(file_name)
        else:
            print('{} is zero length, so ignoring'.format(file_name))

    n_files = len(all_files)
    split_point = int(n_files * SPLIT_SIZE)

    shuffled = random.sample(all_files, n_files)

    train_set = shuffled[:split_point]
    test_set = shuffled[split_point:]

    for file_name in train_set:
        copyfile(SOURCE + file_name, TRAINING + file_name)

    for file_name in test_set:
        copyfile(SOURCE + file_name, TESTING + file_name)


SOURCE_DIR = "Tomato/Tomato_Late_blight/"
TRAINING_DIR = "leaf_disease/training/Tomato_Late_blight/"
TESTING_DIR ="leaf_disease/testing/Tomato_Late_blight/"
split_size = .7
split_data(SOURCE_DIR, TRAINING_DIR, TESTING_DIR, split_size)
```

```
split_data(SOURCE_DIR, TRAINING_DIR, TESTING_DIR, split_size)
```

```
SOURCE_DIR = "Tomato/Tomato_Bacterial_spot/"
TRAINING_DIR = "leaf_disease/training/Tomato_Bacterial_spot/"
TESTING_DIR ="leaf_disease/testing/Tomato_Bacterial_spot/"
split_size = .7
split_data(SOURCE_DIR, TRAINING_DIR, TESTING_DIR, split_size)
```

```
SOURCE_DIR = "Tomato/Tomato_Early_blight/"
TRAINING_DIR = "leaf_disease/training/Tomato_Early_blight/"
TESTING_DIR ="leaf_disease/testing/Tomato_Early_blight/"
split_size = .7
split_data(SOURCE_DIR, TRAINING_DIR, TESTING_DIR, split_size)
```

```
SOURCE_DIR = "Tomato/Tomato_healthy/"
TRAINING_DIR = "leaf_disease/training/Tomato_healthy/"
TESTING_DIR ="leaf_disease/testing/Tomato_healthy/"
split_size = .7
split_data(SOURCE_DIR, TRAINING_DIR, TESTING_DIR, split_size)
```

```
SOURCE_DIR = "Tomato/Tomato_Yellow_Leaf_Curl_Virus/"
TRAINING_DIR = "leaf_disease/training/Tomato_Yellow_Leaf_Curl_Virus/"
TESTING_DIR ="leaf_disease/testing/Tomato_Yellow_Leaf_Curl_Virus/"
split_size = .7
split_data(SOURCE_DIR, TRAINING_DIR, TESTING_DIR, split_size)
```

```
SOURCE_DIR = "Tomato/Tomato_Leaf_Mold/"
TRAINING_DIR = "leaf_disease/training/Tomato_Leaf_Mold/"
TESTING_DIR ="leaf_disease/testing/Tomato_Leaf_Mold/"
split_size = .7
split_data(SOURCE_DIR, TRAINING_DIR, TESTING_DIR, split_size)
```

```
data_dir = "leaf_disease/training"
classes = os.listdir(data_dir)
classes
```

```
['Tomato_Yellow_Leaf_Curl_Virus',
 'Tomato_healthy',
 'Tomato_Bacterial_spot',
 'Tomato_Late_blight',
 'Tomato_Early_blight',
 'Tomato_Leaf_Mold']
```

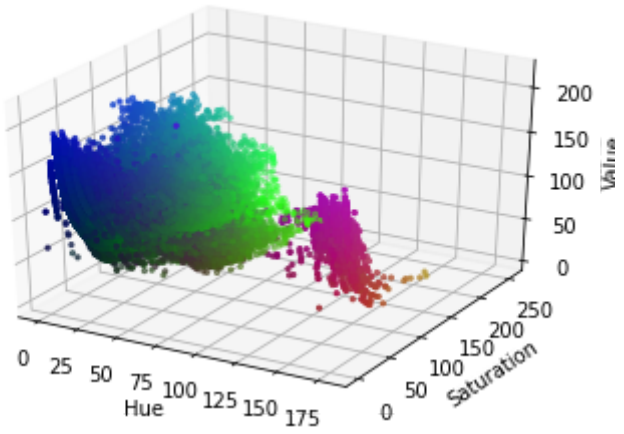
```
img =  cv2.imread('/content/leaf_disease/testing/Tomato_Bacterial_spot/image (100).JPG
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib import colors


pixel_colors = hsv.reshape((np.shape(hsv)[0]*np.shape(hsv)[1], 3))
norm = colors.Normalize(vmin=-1.,vmax=1.)
norm.autoscale(pixel_colors)
pixel_colors = norm(pixel_colors).tolist()
```

```
h, s, v = cv2.split(hsv)
fig = plt.figure()
axis = fig.add_subplot(1, 1, 1, projection="3d")

axis.scatter(h.flatten(), s.flatten(), v.flatten(), facecolors=pixel_colors, marker="
axis.set_xlabel("Hue")
axis.set_ylabel("Saturation")
axis.set_zlabel("Value")
plt.show()
```




```
from google.colab.patches import cv2_imshow
img =  cv2.imread('/content/leaf_disease/testing/Tomato_Bacterial_spot/image (100).JPG
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# find the green color
mask_green = cv2.inRange(hsv, (36,0,0), (86,255,255))
# find the brown color
mask_brown = cv2.inRange(hsv, (8, 60, 20), (30, 255, 200))
# find the yellow color in the leaf
mask_yellow = cv2.inRange(hsv, (21, 39, 64), (40, 255, 255))

# find any of the three colors(green or brown or yellow) in the image
mask = cv2.bitwise_or(mask_green, mask_brown)
mask = cv2.bitwise_or(mask, mask_yellow)
```

```
mask = cv2.bitwise_or(mask, mask_yellow)

# Bitwise-AND mask and original image
res = cv2.bitwise_and(img,img, mask= mask)

cv2_imshow(img)
cv2_imshow(res)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
from google.colab.patches import cv2_imshow
img =  cv2.imread('/content/leaf_disease/testing/Tomato_Yellow_Leaf_Curl_Virus/image (
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

# find the green color
mask_green = cv2.inRange(hsv, (36,0,0), (86,255,255))
# find the brown color
mask_brown = cv2.inRange(hsv, (8, 60, 20), (30, 255, 200))
# find the yellow color in the leaf
mask_yellow = cv2.inRange(hsv, (21, 39, 64), (40, 255, 255))

# find any of the three colors(green or brown or yellow) in the image
mask = cv2.bitwise_or(mask_green, mask_brown)
mask = cv2.bitwise_or(mask, mask_yellow)
```

```
mask = cv2.bitwise_or(mask, mask_yellow)

# Bitwise-AND mask and original image
res = cv2.bitwise_and(img,img, mask= mask)

cv2_imshow(img)
cv2_imshow(res)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   fill_mode='nearest')

valid_datagen = ImageDataGenerator(rescale=1./255)

batch_size = 128
training_dir = 'leaf_disease/training'
```

```
validation_dir = 'leaf_disease/testing'

training_generator = train_datagen.flow_from_directory(training_dir,
                                                       target_size=(256, 256),
                                                       batch_size=batch_size,
                                                       class_mode='categorical')

valid_generator = valid_datagen.flow_from_directory(validation_dir,
                                                    target_size=(256, 256),
                                                    batch_size=batch_size,
                                                    class_mode='categorical',
                                                    shuffle=False)
```

```
Found 8477 images belonging to 6 classes.
Found 6220 images belonging to 6 classes.
```

```
class_dict = training_generator.class_indices
print(class_dict)
```

```
{'Tomato_Bacterial_spot': 0, 'Tomato_Early_blight': 1, 'Tomato_Late_blight': 2,
```

```
target_names = list(class_dict.keys())
print(target_names)
```

```
['Tomato_Bacterial_spot', 'Tomato_Early_blight', 'Tomato_Late_blight', 'Tomato_L
```

```
train_num = training_generator.samples
valid_num = valid_generator.samples
```

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(256, 256, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(6, activation='softmax')
])
```

```
])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 254, 254, 16)      448       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 16)      0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 125, 125, 32)      4640      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 32)        9248      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 64)        18496     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 64)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 64)        36928     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 64)          0         
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0         
_________________________________________________________________
dense (Dense)                (None, 512)               1180160   
_________________________________________________________________
dropout (Dropout)            (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 3078      
=================================================================
Total params: 1,252,998
Trainable params: 1,252,998
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer=Adam(),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(training_generator,
                    steps_per_epoch=train_num//batch_size,
                    validation_data=valid_generator,
                    epochs=47,
                    validation_steps=valid_num//batch_size,
                    )
```

```
Epoch 1/47
67/67 [==============================] - 496s 7s/step - loss: 1.5206 - accuracy:
```

```
Epoch 2/47
67/67 [==============================] - 472s 7s/step - loss: 0.7929 - accuracy:
Epoch 3/47
67/67 [==============================] - 469s 7s/step - loss: 0.6448 - accuracy:
Epoch 4/47
67/67 [==============================] - 448s 7s/step - loss: 0.5230 - accuracy:
Epoch 5/47
67/67 [==============================] - 306s 5s/step - loss: 0.4176 - accuracy:
Epoch 6/47
67/67 [==============================] - 289s 4s/step - loss: 0.3649 - accuracy:
Epoch 7/47
67/67 [==============================] - 289s 4s/step - loss: 0.3369 - accuracy:
Epoch 8/47
67/67 [==============================] - 290s 4s/step - loss: 0.3015 - accuracy:
Epoch 9/47
67/67 [==============================] - 291s 4s/step - loss: 0.2717 - accuracy:
Epoch 10/47
67/67 [==============================] - 290s 4s/step - loss: 0.2397 - accuracy:
Epoch 11/47
67/67 [==============================] - 291s 4s/step - loss: 0.2262 - accuracy:
Epoch 12/47
67/67 [==============================] - 289s 4s/step - loss: 0.2298 - accuracy:
Epoch 13/47
67/67 [==============================] - 289s 4s/step - loss: 0.1690 - accuracy:
Epoch 14/47
67/67 [==============================] - 292s 4s/step - loss: 0.1656 - accuracy:
Epoch 15/47
67/67 [==============================] - 291s 4s/step - loss: 0.1578 - accuracy:
Epoch 16/47
67/67 [==============================] - 295s 4s/step - loss: 0.1729 - accuracy:
Epoch 17/47
67/67 [==============================] - 292s 4s/step - loss: 0.1222 - accuracy:
Epoch 18/47
67/67 [==============================] - 291s 4s/step - loss: 0.1437 - accuracy:
Epoch 19/47
67/67 [==============================] - 289s 4s/step - loss: 0.1324 - accuracy:
Epoch 20/47
67/67 [==============================] - 290s 4s/step - loss: 0.1178 - accuracy:
Epoch 21/47
67/67 [==============================] - 291s 4s/step - loss: 0.1205 - accuracy:
Epoch 22/47
67/67 [==============================] - 289s 4s/step - loss: 0.1169 - accuracy:
Epoch 23/47
67/67 [==============================] - 291s 4s/step - loss: 0.0957 - accuracy:
Epoch 24/47
67/67 [==============================] - 291s 4s/step - loss: 0.0904 - accuracy:
Epoch 25/47
67/67 [==============================] - 289s 4s/step - loss: 0.0891 - accuracy:
Epoch 26/47
67/67 [==============================] - 290s 4s/step - loss: 0.0866 - accuracy:
Epoch 27/47
67/67 [==============================] - 290s 4s/step - loss: 0.0822 - accuracy:
Epoch 28/47
67/67 [==============================] - 290s 4s/step - loss: 0.0885 - accuracy:
Epoch 29/47
67/67 [==============================] - 291s 4s/step - loss: 0.0711 - accuracy:
Epoch 30/47
```

```
model.save('trained_model.h5')


image_path = "image (14).JPG"
new_img=image.load_img(image_path, target_size=(256,256))
img= image.img_to_array(new_img)
img=np.expand_dims(img, axis=0)
img = img/255

img_class = model.predict_classes(img)
prediction = model.predict(img)
print(img_class)
d = prediction.flatten()
j = d.max()
for index,item in enumerate(d):
    if item == j:
        class_name = target_names[index]
print(class_name)
```

```
[1]
Tomato_Early_blight
```

```
image_path = "image (1289).JPG"
new_img=image.load_img(image_path, target_size=(256,256))
img= image.img_to_array(new_img)
img=np.expand_dims(img, axis=0)
img = img/255

img_class = model.predict_classes(img)
prediction = model.predict(img)
print(img_class)
d = prediction.flatten()
j = d.max()
for index,item in enumerate(d):
    if item == j:
        class_name = target_names[index]
print(class_name)
```

```
[4]
Tomato_Yellow_Leaf_Curl_Virus
```

```
history = history
print(history.history.keys())
```

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

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

```
epochs = range(1, len(loss) + 1)

#accuracy plot
plt.plot(epochs, acc, label='Training Accuracy')
plt.plot(epochs, val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.savefig(fname='Training_Validation_Accuracy')
plt.legend()

plt.figure()
#loss plot
plt.plot(epochs, loss, label='Training Loss')
plt.plot(epochs, val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.savefig(fname='Training_Validation_Loss')
plt.show()
```

Training and Validation Accuracy

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import roc_auc_score
from sklearn import preprocessing
```

```
Y_pred = model.predict(valid_generator, valid_num // batch_size+1)
y_pred = np.argmax(Y_pred, axis=1)
print('Confusion Matrix')
print(confusion_matrix(valid_generator.classes, y_pred))
print('Classification Report')
target_names = ['Tomato_Bacterial_spot', 'Tomato_Early_blight',
                'Tomato_Late_blight', 'Tomato_Leaf_Mold',
                'Tomato_Yellow_Leaf_Curl_Virus', 'Tomato_healthy']
print(classification_report(valid_generator.classes, y_pred,
                           target_names=target_names))
```

```
Confusion Matrix
[[762  14   1   0   0   0]
 [  0 361   1   0   0   2]
 [  1  28 645   1   0   4]
 [  0   7   0 333   0   0]
 [  9   4   0   0 483   0]
 [  0   0   0   0   0 588]]
Classification Report
                                precision    recall  f1-score   support

          Tomato_Bacterial_spot       0.99      0.98      0.98       777
            Tomato_Early_blight       0.87      0.99      0.93       364
             Tomato_Late_blight       1.00      0.95      0.97       679
               Tomato_Leaf_Mold       1.00      0.98      0.99       340
Tomato_Yellow_Leaf_Curl_Virus       1.00      0.97      0.99       496
                Tomato_healthy       0.99      1.00      0.99       588

                      accuracy                           0.98      3244
                     macro avg       0.97      0.98      0.98      3244
                  weighted avg       0.98      0.98      0.98      3244
```

```
all_labels = ['Tomato_Bacterial_spot', 'Tomato_Early_blight', 'Tomato_Late_blight',
       'Tomato_Leaf_Mold', 'Tomato_Yellow_Leaf_Curl_Virus', 'Tomato_healthy']
```

```
fig, c_ax = plt.subplots(1,1, figsize = (12, 8))
def multiclass_roc_auc_score(y_test, y_pred, average="macro"):
    lb = preprocessing.LabelBinarizer()
    #lb = LabelBinarizer()
    lb.fit(y_test)
    y_test = lb.transform(y_test)
    y_pred = lb.transform(y_pred)
```

```
    for (idx, c_label) in enumerate(all_labels): # all_labels: no of the labels
        fpr, tpr, thresholds = roc_curve(y_test[:,idx].astype(int), y_pred[:,idx])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.legend(loc = 'lower right')
        plt.title('Receiver Operating Characteristic')
        plt.plot(fpr, tpr, label = '%s (AUC:%0.2f)'  % (c_label, auc(fpr, tpr)))
    plt.plot(fpr, fpr, 'b-', label = 'Random Guessing')
    plt.savefig(fname='roc_auc')
    return roc_auc_score(y_test, y_pred, average=average)

# calling
valid_generator.reset() # resetting generator
y_pred = model.predict(valid_generator, verbose = True)
y_pred = np.argmax(y_pred, axis=1)
multiclass_roc_auc_score(valid_generator.classes, y_pred)
```

```
26/26 [==============================] - 21s 791ms/step
No handles with labels found to put in legend.
0.9874785226766131
```