

```
In [ ]: # _____post training quantization_____
```

```
In [2]: import tensorflow as tf
```

```
C:\Users\heman\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.4)
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}")
```

```
In [4]: # without quantization
convertor = tf.lite.TFLiteConverter.from_saved_model("../leaf disease prediction/Models/quantize_aware_model")
tf_lite_model = convertor.convert()
```

```
In [5]: len(tf_lite_model)
```

```
Out[5]: 743284
```

```
In [ ]:
```

```
In [6]: # with quantization
convertor = tf.lite.TFLiteConverter.from_saved_model("../leaf disease prediction/Models/quantize_aware_model")
convertor.optimizations = [tf.lite.Optimize.DEFAULT]
tf_model1 = convertor.convert()
```

```
In [7]: len(tf_model1)
```

```
Out[7]: 200488
```

```
In [4]: with open("quantize_tflite_model.tflite", "wb") as f:
        f.write(tf_model1)
```

```
In [2]: import os
```

```
In [3]: # creating labels
def recreate_labels():
    # 1) We use this in order to ignore any hidden files that might be here.
    # 'Datasets' is the name of the folder where we store our training data. The 'labels' folder is created here.
    labels = [folder for folder in os.listdir('PlantVillage/Plant') if not folder.startswith('.')]

    # 2) Then, we output the contents of each folder name to a file.
    with open('labels.txt', 'w') as file:
        for label in labels:
            file.write(label + '\n')

recreate_labels()
```

```
In [ ]:
```

```
In [45]: from tensorflow.keras.models import load_model
model = load_model("../leaf disease prediction/Models/quantize_aware_model")
```

```
In [8]: dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "PlantVillage/Plant",
    shuffle = True,
    image_size = (256,256),
    batch_size = 50
)
```

Found 9416 files belonging to 10 classes.

```
In [9]: def get_partition_and_data_tuning(ds,train_split= 0.8 ,val_split=0.1,test_split = 0
    train_size = int(len(dataset)*train_split)
    val_size = int(len(dataset)*val_split)
    test_size = int(len(dataset)*test_split)
    if shuffle:
        ds = ds.shuffle(shuffle_size,seed = 12)
    train_data = ds.take(train_size)
    remain_data = ds.skip(train_size)
    val_data = remain_data.take(val_size)
    test_data = remain_data.skip(val_size)
    train_data = train_data.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
    val_data = val_data.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
    test_data = test_data.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
    return train_data,val_data,test_data
```

```
In [10]: train,val,test = get_partition_and_data_tuning(dataset)
```

```
In [46]: import numpy as np
    def representative_dataset_gen():
        # Provide sample representative input data
        for _ in range(100):
            sample_data = np.random.rand(1, 256, 256, 3)
            yield [sample_data.astype(np.float32)]
```

```
In [47]: converter = tf.lite.TFLiteConverter.from_keras_model(model)
    converter.optimizations = [tf.lite.Optimize.DEFAULT]
    converter.representative_dataset = representative_dataset_gen
    converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
    converter.inference_input_type = tf.uint8
```

```
In [2]: # quantized_tflite_model = converter.convert()
```

```
In [26]: with open('quantized_model.tflite', 'wb') as f:
    f.write(quantized_tflite_model)
```

```
In [1]: # _____--testing the quantize_aware_model_____
```

```
In [3]: interpreter = tf.lite.Interpreter(model_path="quantize_tflite_model.tflite")
```

```
In [6]: input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    print("Input shape:",input_details[0]['shape'])
    print("Input size:",input_details[0]['dtype'])
    print("output shape:",output_details[0]['shape'])
```

```
print("output size:",output_details[0]['dtype'])
interpreter.resize_tensor_input(input_details[0]["index"],(1,256,256,3))
interpreter.resize_tensor_input(output_details[0]["index"],(1,10))
```

Input shape: [50 256 256 3]

Input size: <class 'numpy.float32'>

output shape: [50 10]

output size: <class 'numpy.float32'>

Resize Tensor shape

In [139... `import matplotlib.pyplot as plt`

In [143... `for images_batch,labels_batch in test.take(1):`
`first_img =np.array(images_batch,dtype="float32")`
`# print(first_img)`
`first_label = labels_batch.numpy()`
`interpreter.allocate_tensors()`
`# print("first image to predict")`
`# plt.imshow(first_img)`
`# print("actual label:",dataset.class_names[first_label])`
`interpreter.set_tensor(input_details[0]['index'],first_img)`
`interpreter.invoke()`
`batch_prediction=interpreter.get_tensor(output_details[0]['index'])`
`label1 = np.argmax(batch_prediction)`
`# print("predicted label:",dataset.class_names[label1])`

In [144... `arr = []`

In [145... `for i in range(0,50):`
`arr.append(np.argmax(batch_prediction[i]))`

In [146... `print(arr)`

```
[4, 5, 8, 6, 3, 4, 9, 4, 0, 8, 4, 8, 4, 6, 5, 9, 1, 6, 4, 9, 3, 0, 3, 3, 4, 6, 4, 6,
4, 1, 1, 5, 8, 7, 6, 1, 8, 6, 4, 3, 6, 5, 9, 5, 1, 4, 6, 9, 8, 8]
```

In [147... `first_label`

Out[147... `array([4, 5, 8, 6, 3, 4, 9, 4, 0, 8, 4, 8, 4, 6, 5, 9, 1, 6, 4, 9, 3, 0,`
`3, 3, 4, 6, 4, 6, 4, 1, 1, 5, 8, 7, 6, 1, 8, 6, 4, 3, 6, 4, 9, 5,`
`1, 4, 6, 9, 8, 8])`

In [31]: `for images_batch,labels_batch in test.take(1):`
`first_img = np.array(images_batch[0],dtype="float32")`
`first_label = labels_batch[0].numpy()`
`print(first_img)`
`arr=[first_img]`

```

[[[181. 178. 189.]
  [180. 177. 188.]
  [179. 176. 187.]
  ...
  [195. 190. 197.]
  [186. 181. 188.]
  [202. 197. 204.]]]

[[[176. 173. 184.]
  [176. 173. 184.]
  [176. 173. 184.]
  ...
  [134. 129. 136.]
  [152. 147. 154.]
  [150. 145. 152.]]]

[[[177. 174. 185.]
  [178. 175. 186.]
  [179. 176. 187.]
  ...
  [194. 189. 196.]
  [110. 105. 112.]
  [197. 192. 199.]]]

...

[[[149. 146. 157.]
  [151. 148. 159.]
  [153. 150. 161.]
  ...
  [134. 129. 136.]
  [107. 102. 109.]
  [ 89.  84.  91.]]]

[[[145. 142. 153.]
  [147. 144. 155.]
  [150. 147. 158.]
  ...
  [152. 147. 154.]
  [148. 143. 150.]
  [120. 115. 122.]]]

[[[150. 147. 158.]
  [148. 145. 156.]
  [148. 145. 156.]
  ...
  [121. 116. 123.]
  [108. 103. 110.]
  [144. 139. 146.]]]

```

```

In [32]: interpreter.allocate_tensors()
interpreter.set_tensor(input_details[0]['index'],arr)
interpreter.invoke()
batch_prediction=interpreter.get_tensor(output_details[0]['index'])
label1 = np.argmax(batch_prediction)

```

```
In [33]: print(first_label,label1)
```

8 8

```
In [4]: # %pip install playwright
```

```
In [ ]:
```