Assignment 3 (key)

1. Enable GPU:

```
import tensorflow as tf

# display tf version and test if GPU is active
tf.__version__, tf.test.gpu_device_name()
```

```
('2.3.0', '/device:GPU:0')
```

2. List available TFDS:

```
import tensorflow_datasets as tfds

tfds.list_builders()
```

```
['abstract_reasoning',
 'accentdb',
 'aeslc',
 'aflw2k3d',
```

3. Load **beans** TFDS and inspect:

   a. Load train data with info object and display info:

```
train, info = tfds.load('beans', split='train', with_info=True)
info
```

   b. display train shape:

```
train
```

```
<PrefetchDataset shapes: {image: (500, 500, 3), label: ()}, types: {image: tf.uint8, label: tf.int64}>
```

   c. display number of classes and class labels:

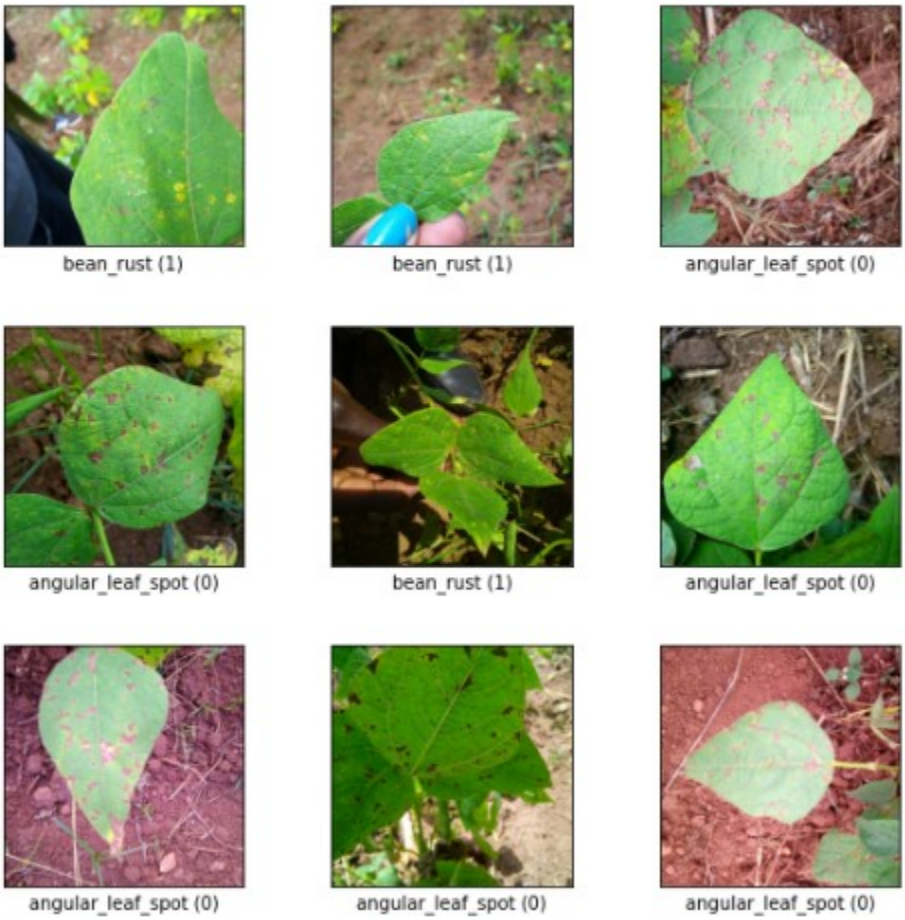
```
num_classes = info.features['label'].num_classes
class_labels = info.features['label'].names

print ('number of classes:', num_classes)
print ('class labels:', class_labels)
```

```
number of classes: 3
class labels: ['angular_leaf_spot', 'bean_rust', 'healthy']
```

   d. display some image examples:

```
fig = tfds.show_examples(train, info)
```



e. display features from FeaturesDict:

```
for examples in train.take(1):
  print(list(examples.keys()))
```

```
['image', 'label']
```

f. display feature shape and target value from the first example:

```
for examples in train.take(1):
  print ('feature shape:', examples['image'].shape)
  print ('target value: ', examples['label'].numpy())
```

```
feature shape: (500, 500, 3)
target value:  1
```

g. display the first 20 labels as numpy:

```
n, ls = 20, []
for examples in train.take(n):
  ls.append(examples['label'].numpy())
ls
```

```
[1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 2, 2, 1, 1, 0, 1, 2]
```

h. display the feature (class) names:

```
names = info.features['label'].names
names
```

```
['angular_leaf_spot', 'bean_rust', 'healthy']
```

i. display the first 20 labels with their class names:

```
for e in ls:
  print (names[e])
```

```
bean_rust
bean_rust
angular_leaf_spot
angular_leaf_spot
bean_rust
angular_leaf_spot
angular_leaf_spot
angular_leaf_spot
angular_leaf_spot
bean_rust
angular_leaf_spot
angular_leaf_spot
angular_leaf_spot
healthy
healthy
bean_rust
bean_rust
angular_leaf_spot
bean_rust
healthy
```

j. display number of train, test, and validation examples:

```
train_examples = info.splits['train'].num_examples
test_examples = info.splits['test'].num_examples
valid_examples = info.splits['validation'].num_examples
train_examples, test_examples, valid_examples
```

k. display splits:

```
info.splits
```

```
{'test': <tfds.core.SplitInfo num_examples=128>,
 'train': <tfds.core.SplitInfo num_examples=1034>,
 'validation': <tfds.core.SplitInfo num_examples=133>}
```

l. display name of dataset:

```
info.name
```

```
'beans'
```

m. display supervised keys:

```
info.supervised_keys
```

```
('image', 'label')
```

(hint: to see available metadata object, type **info.**)

4. Load **beans** TFDS with **as_supervised=True**, scale, and build input pipeline.

   a. load train, test, and validation sets:

```
train= tfds.load('beans', split='train', as_supervised=True)
test = tfds.load('beans', split='test', as_supervised=True)
validation = tfds.load('beans', split='validation', as_supervised=True)
```

   b. create scale function (function included for your convenience):

```
def scale(image, label):
  image = tf.cast(image, tf.float32)
  image /= 255
  return image, label
```

   c. finish pipeline (code included for your convenience):

```
train_ds = train.map(scale).cache().shuffle(100).batch(32)
test_ds = test.map(scale).batch(32)
valid_ds = validation.map(scale).batch(32)
```

5. Model.

   a. import libraries:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
```

b. get input shape:

```
for img, lbl in train.take(1):
  in_shape = img.shape
in_shape
```

```
TensorShape([500, 500, 3])
```

c. create model:

```
# clear previous model
tf.keras.backend.clear_session()

model = Sequential([
  Flatten(input_shape=in_shape),
  Dense(512, activation='relu'),
  Dense(10, activation='softmax')
])
```

(hint: change the input shape to reflect bean images)

d. compile:

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

e. train for 3 epochs:

```
epochs = 3
history = model.fit(train_ds, epochs=epochs, verbose=1,
                    validation_data=test_ds)
```

(note: feedforward network is not ideal for complex images)

f. generalize:

```
model.evaluate(test_ds)
```

```
4/4 [==============================] - 0s 49ms/step - loss: 175.4465 - accuracy: 0.4609
[175.446533203125, 0.4609375]
```

6. Predict.

a. get predictions (use **valid_ds**):

```
predictions = model.predict(valid_ds)
```

b. display prediction array for first element in dataset:

```
predictions[0].round(2)
```

```
array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

(note: predictions may vary due to randomness)

c. display prediction for first element:

```
import numpy as np

first_pred = np.argmax(predictions[0]) # start at index 0
print ('predicted:', first_pred)
```

```
predicted: 1
```

(note: predictions may vary due to randomness)

d. display prediction by class name:

```
names[first_pred]
```

```
'bean_rust'
```

(note: predictions may vary due to randomness)

e. display actual by class name:

```
for feature, label in valid_ds.take(1):
  print ('label:', names[label[0]])
```

```
label: bean_rust
```

(note: predictions may vary due to randomness)

f. display predictions for the first batch:

```
p32 = tf.convert_to_tensor([np.argmax(predictions[p]) for p in range(32)])
p32
```

```
<tf.Tensor: shape=(32,), dtype=int32, numpy=
array([1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 1, 2, 2, 1, 2, 1, 1, 1, 1, 2, 2,
       2, 1, 2, 1, 1, 2, 2, 1, 1, 1], dtype=int32)>
```

(hint: batch size is 32)

g. display actual labels for the first batch:

```
for feature, label in valid_ds.take(1):
  print ('label:', label)
```

```
label: tf.Tensor([1 1 1 1 1 1 1 2 2 0 2 2 2 2 0 2 1 1 0 1 0 2 0 0 2 1 0 2 2 1 0 0], shape=(32,), dtype=int64)
```

h. display number and percentage of misclassifications for the first batch:

```
cnt = 0
for i in range(32):
  if p32[i].numpy() != label[i].numpy():
    cnt += 1
print (cnt)
perc = str((cnt / 32) * 100) + '%'
perc
```

```
10
'31.25%'
```