Rock, Paper, Scissors

In this week's exercise, you will use TFDS module for performing extract, transform and load (ETL) tasks on the Rock-Paper-Scissors dataset.

Upon completion of the exercise, you will have

- Loaded the dataset
- Transformed and preprocessed it
- Defined a simple CNN model for image classification which can be trained easily

Step 0 - Setup

Note : The assignment uses TF version 2 so if you run this notebook on TF 1.x, some things might not work.

```
from os import getcwd

import tensorflow as tf
import tensorflow_datasets as tfds
```

Step 1 - One-Hot Encoding

Remember to one hot encode the labels as you have 3 classes - Rock, Paper and Scissors. You can use Tensorflow's one_hot function (`tf.one_hot`) to convert categorical variables to one-hot vectors.

Useful parameters -

1. `indices` - A tensor containing all indices
2. `depth` - A scalar defining the depth of the one hot dimension.

```
# EXERCISE: encoding the labels using your own function for one-hot
encoding

def my_one_hot(feature, label):
    # Encode the labels to one-hot using tf.one_hot with depth equal
to total
    # number of classes here which are rock, paper and scissors

    one_hot = tf.one_hot(label, depth=3)
    return feature, one_hot

# TESTING THE FUNCTION
_,one_hot = my_one_hot(["a","b","c","a"],[1,2,3,1])
print(one_hot)
```

```
tf.Tensor(
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]
 [0. 1. 0.]], shape=(4, 3), dtype=float32)
```

Expected Output

```
tf.Tensor(
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]
 [0. 1. 0.]], shape=(4, 3), dtype=float32)
```

Step 2 - Loading Dataset

You will be using `tfds.load()`] method to load the dataset. The dataset is already downloaded and unzipped for you in the data folder but if you are running on your local machine and do not have the dataset downloaded, it will first download and save the dataset to your tensorflow directory and then load it.

Useful parameters -

1. `split` - Which split of the data to load (e.g. 'train', 'test' ['train', 'test'], 'train[80%:]',...)
2. `data_dir` - Directory to read/write data. Defaults to the value of the environment variable *TFDS_DATA_DIR*, if set, otherwise falls back to "~/tensorflow_datasets"
3. `as_supervised`- If True, the returned tf.data.Dataset will have a 2-tuple structure (input, label) according to builder.info.supervised_keys. If False the default, the returned tf.data.Dataset will have a dictionary with all the features

Note - The`rock_paper_scissors:3.*.*` dataset is already downloaded for you so if you specify the major version thisway while loading, it will try to find the dataset from the directory and load it. If none is present or the current dataset has been upgraded to a new major version, then it will try to download the new dataset to the directory.

```
# EXERCISE: Loading the rock, paper and scissors train and test
dataset using tfds.load.

# Use data_dir=filepath as the dataset is already downloaded for you

filePath = f"{getcwd()}/data"

train_data = tfds.load('rock_paper_scissors', data_dir=filePath,
split='train', as_supervised=True)
val_data = tfds.load('rock_paper_scissors', data_dir=filePath,
split='test', as_supervised=True)
```

```
# Testing train_data and val_data if loaded correctly

train_data_len = len(list(train_data))
val_data_len = len(list(val_data))

print(train_data_len)
print(val_data_len)

2520
372
```

Expected Output

```
2520
372
```

Step 3 - Mapping one hot encode function to dataset

You will apply the `my_one_hot()` encoding function to the train and validation data using `map` function. It will apply the custom function to each element of the dataset and returns a new dataset containing the transformed elements in the same order as they appeared in the input.

```
# EXERCISE: one-hot encode the train and validation labels using the
function you defined earlier

# HINT - use map function
https://www.tensorflow.org/api_docs/python/tf/data/Dataset#map

train_data = train_data.map(my_one_hot)
val_data = val_data.map(my_one_hot)

print(type(train_data))

<class 'tensorflow.python.data.ops.dataset_ops.MapDataset'>
```

Expected Output

```
<class 'tensorflow.python.data.ops.dataset_ops.MapDataset'>
```

Step 4 - Exploring dataset metadata

Do remember that `tfds.load()` has a parameter called `with_info` which if True will return the tuple (tf.data.Dataset, tfds.core.DatasetInfo) containing the info associated with the builder.

```
# EXERCISE: Check the information about the dataset to see the dataset
image shape

# HINT: Use with_info=True and data_dir
_,info = tfds.load('rock_paper_scissors', data_dir=filePath,
with_info=True)
```

```
# DO NOT EDIT THIS
print(info.features['image'].shape)
```

```
(300, 300, 3)
```

Expected Output

```
(300, 300, 3)
```

Step 5 - Training your simple CNN classifier

Now you will define a simple 1-layer CNN model which will learn to classify the images into rock, paper and scissor!

```
# EXERCISE: Train a simple CNN model on the dataset

train_batches = train_data.shuffle(100).batch(10)
validation_batches = val_data.batch(32)

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3, 3), activation='relu',
input_shape=(300, 300, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(3, activation='softmax')
])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 298, 298, 16)      448

 max_pooling2d (MaxPooling2D  (None, 149, 149, 16)     0
 )

 flatten (Flatten)           (None, 355216)            0

 dense (Dense)               (None, 3)                 1065651

=================================================================
Total params: 1,066,099
Trainable params: 1,066,099
Non-trainable params: 0
_________________________________________________________________
```

Submission Instructions

```
# Now click the 'Submit Assignment' button above.
```

[Optional] Step 6 - Evaluation

Remember to submit your assignment before you uncomment and run the next cell.

```
# # OPTIONAL EXERCISE: Compile and fit your model - use categorical
loss and choose optimizer as Adam

# EPOCH = 3

# # You should get decent enough training accuracy in 3-4 epochs
itself as this one layer model will heavily overfit

# model.compile(# YOUR CODE HERE)

# history = model.fit(train_batches, epochs= EPOCH ,
validation_data=validation_batches, validation_steps=1)

# print("Final Training Accuracy:-",history.history['accuracy'][-1])
# print("Final Validation Accuracy:-",history.history['val_accuracy']
[-1])
```

When you're done or would like to take a break, please run the two cells below to save your work and close the Notebook. This frees up resources for your fellow learners.

```
%%javascript
<!-- Save the notebook -->
IPython.notebook.save_checkpoint();

%%javascript
<!-- Shutdown and close the notebook -->
window.onbeforeunload = null
window.close();
IPython.notebook.session.delete();
```