Congratulations! You passed!

Grade received 85.71% To pass 80% or higher

Go to next item

Week 2

Total points 7

1. If you want to merge two splits 'train' and 'test' together using Splits API, how would you be able to do so? **1 / 1 point**

- (●) tfds.load('mnist', split = 'train + test')
- ( ) tfds.load('mnist', split = pd.concat('train', 'test'))
- ( ) tfds.load('mnist', merge = 'train+test')
- ( ) tfds.load('mnist',split = np.concat('train+test'))

Correct
Correct!

Passing both train and test in the string is the proper way to get both the splits.

2. The MNISTv3 dataset supports the Splits API. The train split has 70000 records in it. If you just want to create a subsplit of the first 7000 records and want to use the python slicing notation instead of Splits API, what would be the answer? **1 / 1 point**

- (●) tfds.load('mnist:3.*.*', split='train[:7000]')
- ( ) tfds.load('mnist:3.*.*', split='train[7000:]')
- ( ) tfds.load('mnist:3.*.*', subsplit='train[:7000]')
- ( ) Read the entire train split, create a new dataset, iterate over the first 7000 of the 70000, and copy the records one-by-one to the new dataset.

Correct
Correct!

train[:7000] technically takes records from 0 to 6999 index value.

3. If you want a subsplit of the first 10% of the MNISTv3 training records, what would the code look like using the Splits API? **1 / 1 point**

- (●) tfds.load('mnist:3.*.*', split='train[:10%]')
- ( ) tfds.load('mnist:3.*.*', split='train[10%:]')
- ( ) tfds.load('mnist:3.*.*', subsplit='train[:10%]')
- ( ) tfds.load('mnist:3.*.*', subsplit='train[10%:]')

Correct
Correct!

'train[:10%]' in string format represents that we want the first 10% of the records from the train split.

4. How many validation splits will this code generate? **1 / 1 point**

val_ds = tfds.load('mnist:3.*.*', split = ['train[{}]%:{}%]'.format(k/4,(k+40)/4) for k in range(0,400,40)])

- ( ) 40
- (●) 10
- ( ) 5
- ( ) Will throw an Error

Correct
Correct!

As k is incremented by 40, you get the values like (0,40), (40,80)... until the last one, (360:400).

Dividing each value by 4 as you have (k/4,(k+40)/4), it will get converted to [0,10],[10,20]...[90,100] which is 10 splits.

5. True or False : The TFRecord shards are only created for the training data. **1 / 1 point**

- (●) False
- ( ) True

Correct
If your validation and test data size is bigger, they also can get sharded.

6. Which of the following could be used to investigate how the TFRecords files look like? **0 / 1 point**

- ( ) filename = "your_tf_record_file"raw_file = tf.data.TFRecordDataset(filename)for raw_record in raw_file.take(1): print(repr(raw_record))
- ( ) filename = "your_tf_record_file"raw_file = tf.loads(filename)for raw_record in raw_file.take(1): print(repr(raw_record))
- ( ) filename = "your_tf_record_file"raw_file = tf.keras.dataset.load(filename)for raw_record in raw_file.take(1): print(repr(raw_record))
- (●) filename = "your_tf_record_file"raw_file = tf.RecordDataset(filename)for raw_record in raw_file.take(1): print(repr(raw_record))

Incorrect
TFRecordDataset is part of the tf.Data class. The class tf.RecordDataset does not exist.

7. Below is an example of how raw TFRecord files look like when you properly read and print them with TFRecordDataset methods. **1 / 1 point**

```
<tf.Tensor: shape=(), dtype=string, numpy=b'\nQ\n\x13\n\x08feature2\x12\x07\n\x05\n\x03cat\n\x14
<tf.Tensor: shape=(), dtype=string, numpy=b'\nS\n\x15\n\x08feature2\x12\t\n\x07\n\x05horse\n\x14
<tf.Tensor: shape=(), dtype=string, numpy=b'\nU\n\x11\n\x08feature0\x12\x05\x1a\x03\n\x01\x01\n\
<tf.Tensor: shape=(), dtype=string, numpy=b'\nU\n\x11\n\x08feature0\x12\x05\x1a\x03\n\x01\x00\n\
<tf.Tensor: shape=(), dtype=string, numpy=b'\nR\n\x11\n\x08feature0\x12\x05\x1a\x03\n\x01\x00\n\
<tf.Tensor: shape=(), dtype=string, numpy=b'\nU\n\x14\n\x08feature3\x12\x08\x12\x06\n\x04[\x19\x
<tf.Tensor: shape=(), dtype=string, numpy=b'\nS\n\x15\n\x08feature2\x12\t\n\x07\n\x05horse\n\x14
<tf.Tensor: shape=(), dtype=string, numpy=b'\nQ\n\x13\n\x08feature2\x12\x07\n\x05\n\x03dog\n\x11
<tf.Tensor: shape=(), dtype=string, numpy=b'\nR\n\x11\n\x08feature0\x12\x05\x1a\x03\n\x01\x00\n\
<tf.Tensor: shape=(), dtype=string, numpy=b'\nQ\n\x11\n\x08feature1\x12\x05\x1a\x03\n\x01\x00\n\
```

To parse them into proper format, which of the following options need to be implemented?

- [ ] Creating a parsing function using tfds.load()
- [ ] Apply the parsing function to each item in the dataset using thekeras.dataset.map method
- [x] Apply the parsing function to each item in the dataset using thetf.data.Dataset.mapmethod.

Correct
Correct!

This is Step 3. You need to parse each file individually and map them to the parsing function to create clean, readable standard tensors.

- [x] Creating a feature description dictionary

Correct
Correct!

This is Step 1. You need to define your feature descriptions properly based on the dataset and its metadata.This Is necessary here because datasets use graph-execution, and need description to build their shape and type signature

- [x] Creating a parsing function using tf.io.parse_single_example()

Correct
Correct!

This is Step 2. ***tf.io.parse_single_example()*** is used to parse examples one by one as the raw TFRecordDataset files contain serialized tf.train.Example objects.

You can also use ***tf.parse_example***to parse the whole batch at once.