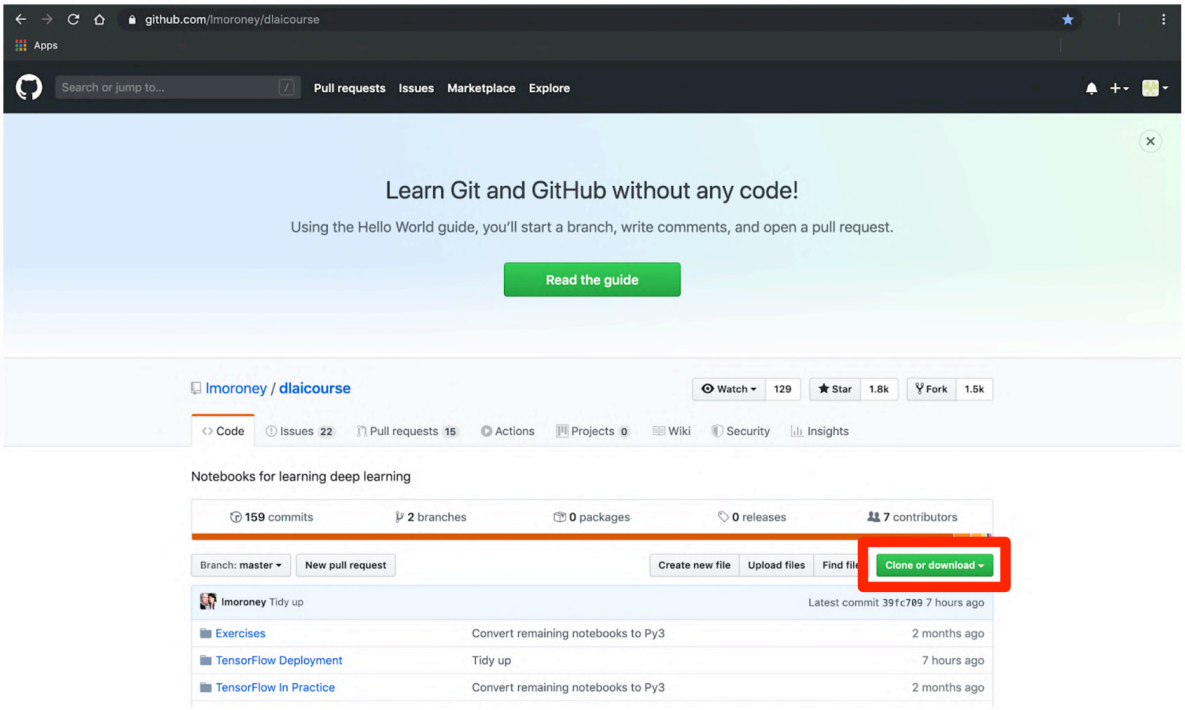
Downloading the Coding Examples and Exercises

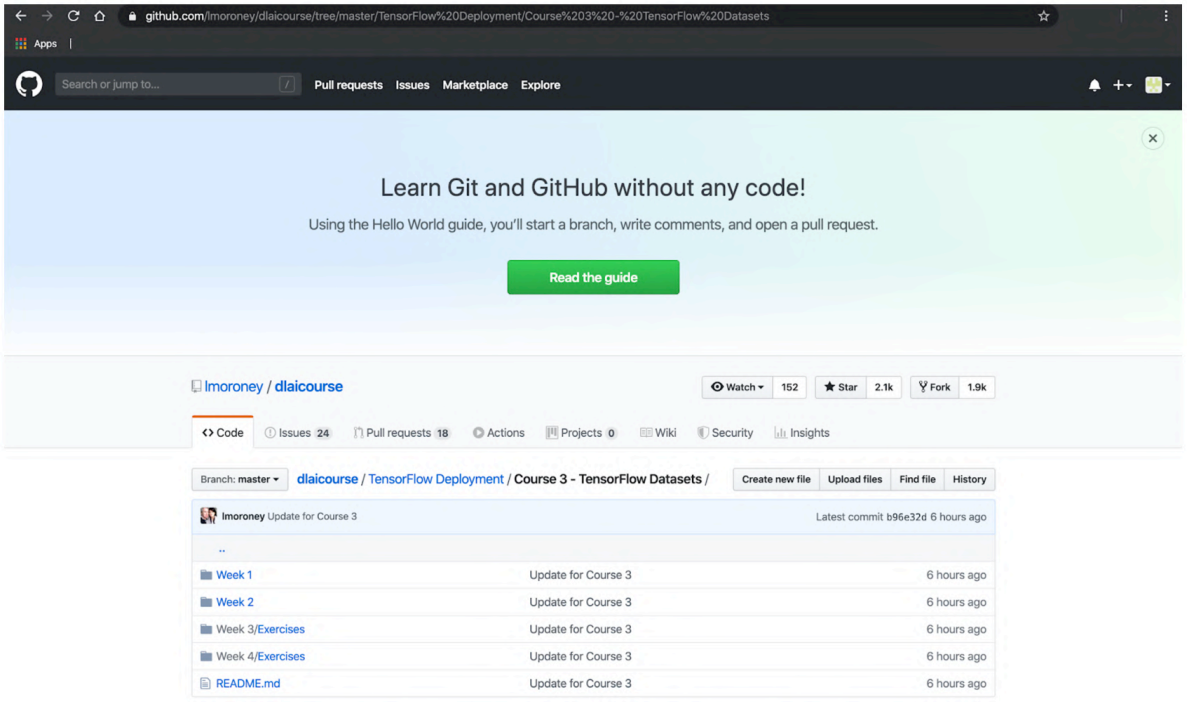
We have created this GitHub Repository where you can find all the examples and exercises not only for this course but for the entire TensorFlow for Data and Deployment Specialization .

You can download all the examples and exercises to your computer by cloning or downloading the GitHub Repository.



You can find the corresponding coding examples and exercises for this course in the following folder in the GitHub repository:

[dlaicourse/TensorFlow Deployment/Course 3 - TensorFlow Datasets/]



Each folder contains the corresponding examples and exercises for each week of this course on TensorFlow Datasets.

NOTE: The code in the repository is updated occasionally. Therefore the code in the repository may vary slightly from the one shown in the videos.

TensorFlow Datasets (TFDS)

- Ready-to-use
- Flexible
- Standardized input pipelines
- Plethora of public research data
- Seamless integration
- Faster prototyping

Some popular datasets

Image
MNIST
CIFAR10
COCO2014
KITTI

Structured
Titanic
IRIS
Amazon US reviews

Text
IMDB reviews
Wikipedia
CNN - Daily Mail
SQuAD

Audio
NSynth
Groove

Video
UCF-101
Moving MNIST

Translate
WMT
TED multi-translate

ETL for TensorFlow

TFDS and tf.data in a nutshell

Typically, the extract phase would involve something like downloading a zip file containing the data, sorting it into directories, loading from those directories, etc. But in this case as the data is already in TFDS, we don't have to bother with all of that.

```
import tensorflow as tf
import tensorflow_datasets as tfds

# Construct a tf.data.Dataset by downloading and extracting
dataset = tfds.load(name="mnist", split="train")

dataset = dataset.shuffle(NUM_SAMPLES) # buffer size
dataset = dataset.repeat(NUM_EPOCHS)
dataset = dataset.map(lambda x: ...)
dataset = dataset.batch(BATCH_SIZE)

iterator = dataset.take(10) # To fetch 10 samples from the dataset
for data in iterator:
    # Access data and use it
```

Extract

Transform

Load

TFDS and tf.data in a nutshell


```
import tensorflow as tf
import tensorflow_datasets as tfds

# Construct a tf.data.Dataset by downloading and extracting
dataset = tfds.load(name="mnist", split="train")

dataset = dataset.shuffle(NUM_SAMPLES) # buffer size
dataset = dataset.repeat(NUM_EPOCHS)
dataset = dataset.map(lambda x: ...)
dataset = dataset.batch(BATCH_SIZE)

iterator = dataset.take(10) # To fetch 10 samples from the dataset
for data in iterator:
    # Access data and use it
```

Extract

Transform

Load

TFDS and tf.data in a nutshell


```
import tensorflow as tf
import tensorflow_datasets as tfds

# Construct a tf.data.Dataset by downloading and extracting
dataset = tfds.load(name="mnist", split="train")

dataset = dataset.shuffle(NUM_SAMPLES) # buffer size
dataset = dataset.repeat(NUM_EPOCHS)
dataset = dataset.map(lambda x: ...)
dataset = dataset.batch(BATCH_SIZE)

iterator = dataset.take(10) # To fetch 10 samples from the dataset
for data in iterator:
    # Access data and use it
```

Extract

Transform

Load

Playing it simple

```
import tensorflow as tf
import tensorflow_datasets as tfds

# Construct a tf.data.Dataset from MNIST
dataset = tfds.load(name="mnist")

>>> dataset
{'train': ..., 'test': ...}
```

What if you wanted to only load the train split of the dataset? You could do this by explicitly specifying the required split you wish to load by making use of the split parameter in the tfds.load API.

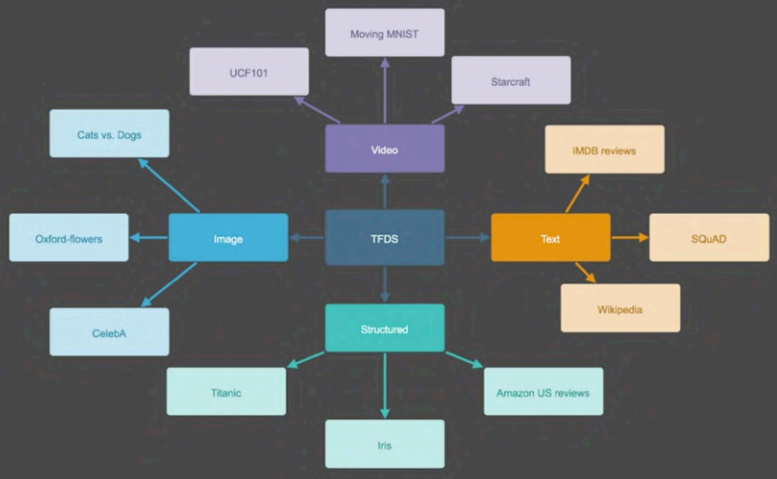
Now, coming back to inspecting the dataset, what you can see is the actual datasets representation in the form of an OptionsDataset object. This object indicates the shapes and types of the inputs and targets of a particular dataset. In this example, we see that the train split has monochrome image data of shape 28 by 28 by 1 in unsigned integers. It also has the associated labels of type int 64 in this case. Finally, we can confirm that the obtained split is an instance of tf.data.Dataset.

Choosing a dataset split

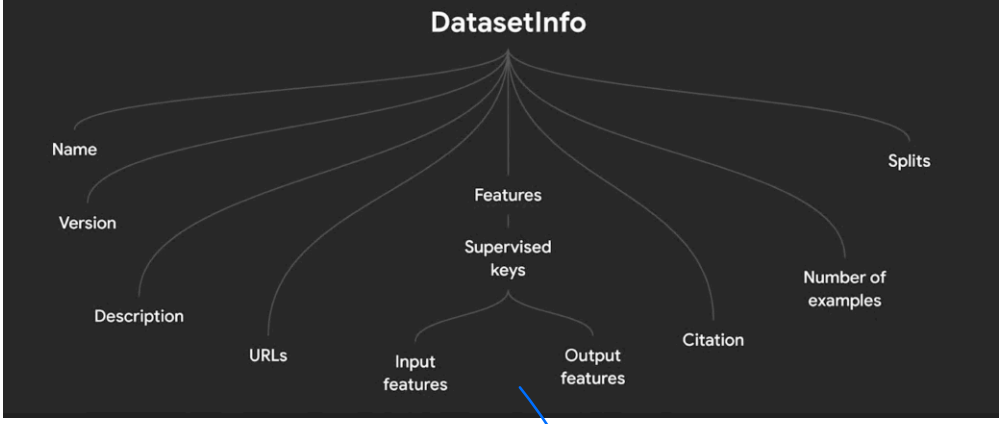
```
# Construct a tf.data.Dataset from MNIST
dataset = tfds.load(name="mnist", split="train")

# Inspecting shapes and datatypes
>>> dataset
{'train': <_OptionsDataset shapes: {image: (28, 28, 1), label: ()},
                           types: {image: tf.uint8, label: tf.int64}>}

# Checking if the dataset is an instance of tf.data.Dataset
>>> assert isinstance(dataset['train'], tf.data.Dataset)
True
```

Listing all datasets

```
# See available datasets
print(tfds.list_builders())
```



Viewing a dataset's metadata




```
mnist, info = tfds.load('mnist', with_info=True)
>>> info

tfds.core.DatasetInfo(
    name='mnist'
    version=1.0.0,
    description='The MNIST database of handwritten digits.',
    urls=['https://storage.googleapis.com/cvdf-datasets/mnist/'],
    features=FeaturesDict({
        'image': Image(shape=(28, 28, 1), dtype=tf.uint8),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=10),
    }),
    total_num_examples=70000,
    splits={'test': 10000, 'train': 60000},
    supervised_keys=('image', 'label'),
    citation="""@article{lecun2010mnist,
      title={MNIST handwritten digit database},
      author={LeCun, Yann and Cortes, Corinna and Burges, CJ},
      journal={ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist},
      volume={2},
      year={2010}
```

We have the feature dictionary comprising of input and output features, as well as the names of the supervised inputs and targets. You can also see how many examples are present in a particular dataset with a total_num_examples property along with the number of samples in each split.

Extracting properties from DatasetInfo

```
>>> print('URLs: ', info.urls)
URLs:  ['https://storage.googleapis.com/cvdf-datasets/mnist/']

>>> print('Image features: ', info.features['image'])
Image features:  Image(shape=(28, 28, 1), dtype=tf.uint8)

>>> print('Label features: ', info.features['label'])
Label features:  ClassLabel(shape=(), dtype=tf.int64, num_classes=10)

>>> print('Number of training examples ', info.splits['train'].num_examples)
Number of training examples  60000

>>> print('Number of test examples ', info.splits['test'].num_examples)
Number of test examples  10000
```

Loading a specific version

If you're told you wish to have the latest version inside a particular major version, specify the desired major version and place asterisks for the rest.

```
mnist = tfds.load("mnist:1.*.*")
```

Major version | Minor version | Patch version

Major version change

If you increment the major version, the slicing API may not necessarily return the same set of records for a given slice. This can even be the case if the existing data has changed.

Minor version change

An increase in the minor version signifies that there's no change in the existing data, but implies that some additional features have been added.

Patch version change

If the patch version has increased, the serialization on disk may have changed.

Loading a dataset (as_supervised=True)

```
dataset = tfds.load('mnist', as_supervised=True)

# Inspecting shapes of a batch
>>> for image, label in dataset['train'].take(1):
        print(image.shape, label.shape)
(1, 28, 28, 1) (1,)
```

A handy feature of loading a dataset is to specify it a supervised. If you do this with as supervised equals true, then your dataset will be preformatted into tuples of data and label as you can see here.

as_supervised=True

Your dataset will be pre-formatted into tuples of data and label.

as_supervised=False

If you set it to false, your dataset will be available as a dictionary.

Using existing splits

Non-conventional named splits

```
split = tfds.Split('test2015')
ds = tfds.load('coco2014', split=split)
```

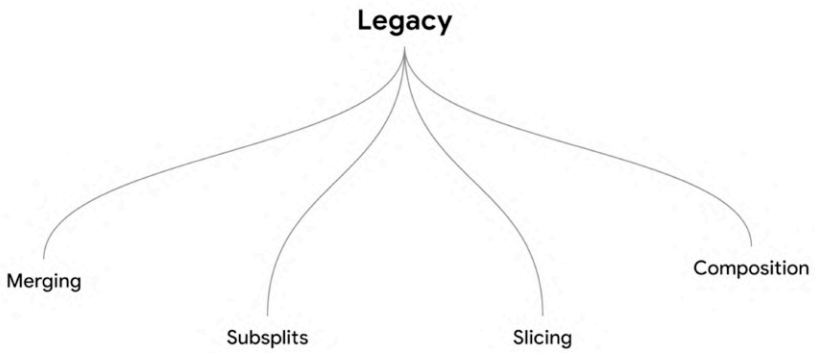
DatasetBuilder

```
mnist_builder = tfds.builder("mnist")

mnist_builder.download_and_prepare()

mnist_builder.as_dataset(split=tfds.Split.TRAIN)
```

Earlier, we mentioned that the extract phase of ETL was done by the TFDS Load method. Under the hood, it goes through several stages to construct a data set builder, and if you prefer, you can explicitly call these stages individually. These may include instantiating a builder for a particular datasets, then downloading and extracting it, and ultimately, setting up the dataset by writing the TF records to disk, which is essential for loading the dataset.

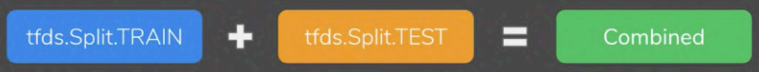
Two APIs

Legacy	Splits API (new)
• Splitting with tfds.Split • Support for all datasets in TFDS • Features slicing	• Slicing with a string expression • Newly added datasets support this



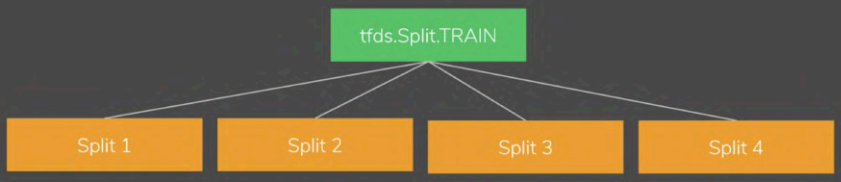
Legacy API

Merging splits



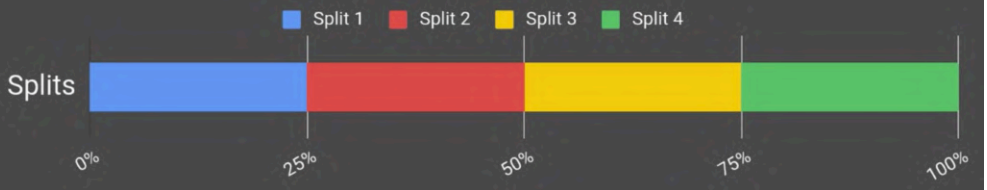

```
all = tfds.Split.TRAIN + tfds.Split.TEST
ds = tfds.load("mnist", split=all)

>>> print(len(list(ds)))
70000
```

Creating subsplits



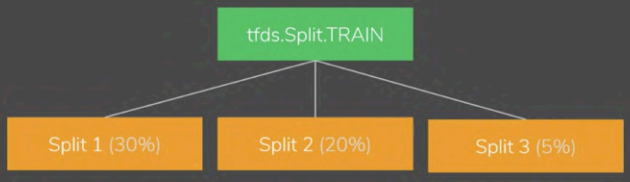

```
s1, s2, s3, s4 = tfds.Split.TRAIN.subsplit(k=4)

dataset_split_1 = tfds.load("mnist", split=s1)
dataset_split_2 = tfds.load("mnist", split=s2)
...
```

Creating subsplits



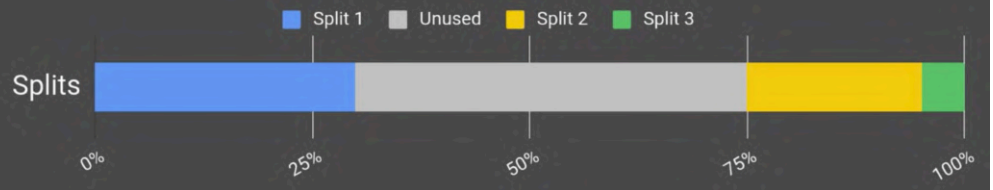

```
NamedSplit('train')(tfds.percent[0:25]) # Split 1
NamedSplit('train')(tfds.percent[25:50]) # Split 2
NamedSplit('train')(tfds.percent[50:75]) # Split 3
NamedSplit('train')(tfds.percent[75:100]) # Split 4
```

Percentage slicing



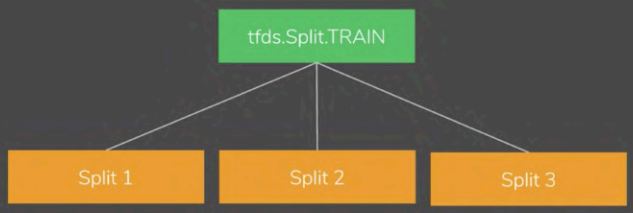

```
first_30_percent = tfds.Split.TRAIN.subsplit(tfds.percent[:30])
last_5_percent = tfds.Split.TRAIN.subsplit(tfds.percent[95:])
middle_20_percent = tfds.Split.TRAIN.subsplit(tfds.percent[75:95])
```

Percentage slicing



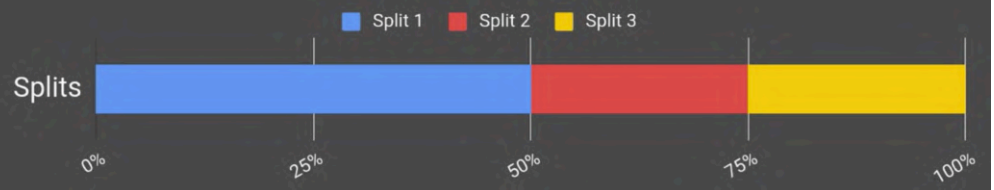

```
NamedSplit('train')(tfds.percent[:30]) # Split 1
NamedSplit('train')(tfds.percent[75:95]) # Split 2
NamedSplit('train')(tfds.percent[95:]) # Split 3
```

Weighted splits



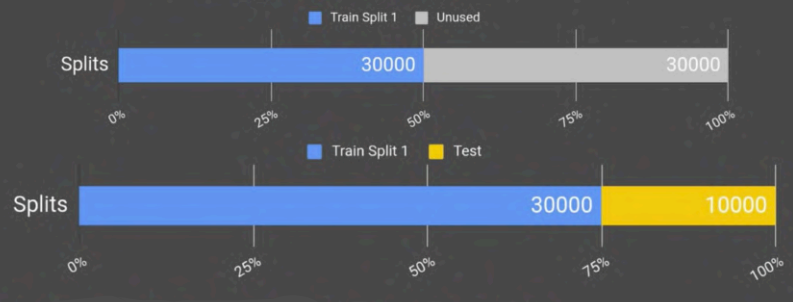

```
half, quarter1, quarter2 = tfds.Split.TRAIN.subsplit(weighted=[2, 1, 1])
```

Weighted splits




```
NamedSplit('train')(tfds.percent[0:50]) # Split 1
NamedSplit('train')(tfds.percent[50:75]) # Split 2
NamedSplit('train')(tfds.percent[75:100]) # Split 3
```

Composing operations



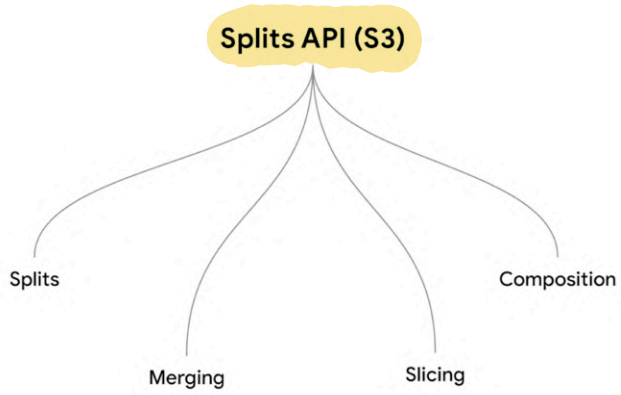
Invalid usages

```
# INVALID! TRAIN included twice
split = tfds.Split.TRAIN.subsplit(tfds.percent[:25]) +
        tfds.Split.TRAIN

# INVALID! Subsplit of subsplit
split = tfds.Split.TRAIN.subsplit(tfds.percent[0:25]).subsplit(k=2)

# INVALID! Subsplit of subsplit
split = (tfds.Split.TRAIN.subsplit(tfds.percent[:25]) +
         tfds.Split.TEST).subsplit(tfds.percent[0:50])
```

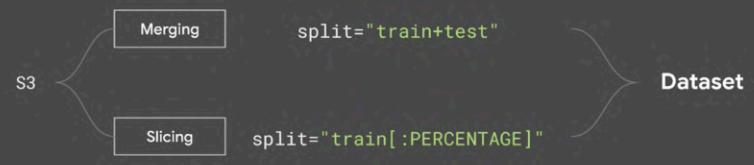
You can't add the same split covering the same segments of the split more than once. You can see this here where we take the first 25% of the set and then add the full set to it, so a segment is repeated

You cannot chain a subsplit operation. For example, here we subsplit the train, and then attempt to subsplit that again.

Finally, you cannot obtain a subsplit of an already merged dataset which you can see in the third example. So there, one of the operations is valid and this involves merging the first 25% of the train split and the entire test split. However, you can't go about merging this with another split by chaining a subsplit operation again.



Splits API



Check dataset for S3 support

```
mnist_builder = tfds.builder("rock_paper_scissors:3.*.*")

>>> mnist_builder.version.implements(tfds.core.Experiment.S3)
True
```

https://www.tensorflow.org/datasets/api_docs/python/tfds/core/Experiment

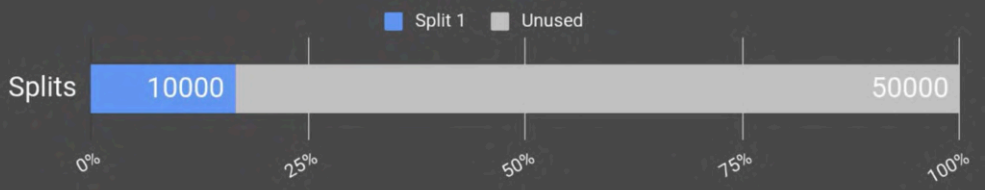
Distinct splits

```
# The full `train` split and the full `test` split as two distinct datasets.
train_ds, test_ds = tfds.load('mnist:3.*.*', split=['train', 'test'])
```

Merging


```
# The full `train` and `test` splits, concatenated together.
combined = tfds.load('mnist:3.*.*', split='train+test')
```

Slicing by index




```
tfds.load('mnist:3.*.*', split='train[:10000]')
```

Slicing by percentage


```
tfds.load('mnist:3.*.*', split='train[:20%]')
```

K-fold splits

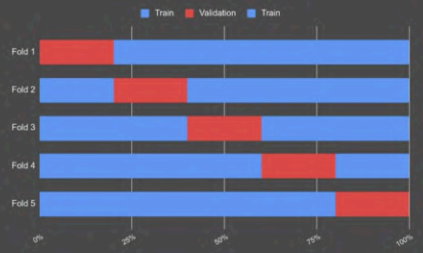
The **validation** datasets are each going to be 20%

`[0%:20%], [20%:40%], ..., [80%:100%]`

The **training** datasets are each going to be the complementary 80%

`[20%:100%]`	(for a validation set of `[0%:20%]`)
`[0%:20%] + [20%:100%]`	(for a validation set of `[20%:40%]`)
`[0%:80%]`	(for a validation set of `[80%:100%]`)

And so on ...




```
val_ds = tfds.load('mnist:3.*.*', split=['train[{}%:{}%]'.format(k, k+20)
                                         for k in range(0, 100, 20)])
train_ds = tfds.load('mnist:3.*.*', split=['train[:{}%]+train[{}%:]'.format(k, k+20)
                                           for k in range(0, 100, 20)])
```

Composing operations


```
# The first 10% of test + the last 80% of train.
10_80pct_ds = tfds.load('mnist:3.*.*', split='test[:10%]+train[-80%:]')
```