Copyright Notice

Image

Classification

Object detection

Segmentation

Generation

Structured

Regression

Recommendation

Text

Classification

Sentiment Analysis

Generation

Question-answering

Audio

Speech recognition

Music recommendation

Audio segmentation

Video

Action recognition

Video classification

Object tracking

VIdeo understanding

Translate

Neural machine translation

Transliteration

Unsupervised Machine Translation

Data is not what it seems

Generated with WCG

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

```
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")
```

Extract

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

Transform

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

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")
```

Extract

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

Transform

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

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")
```

Extract

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

Transform

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

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': ...}
```

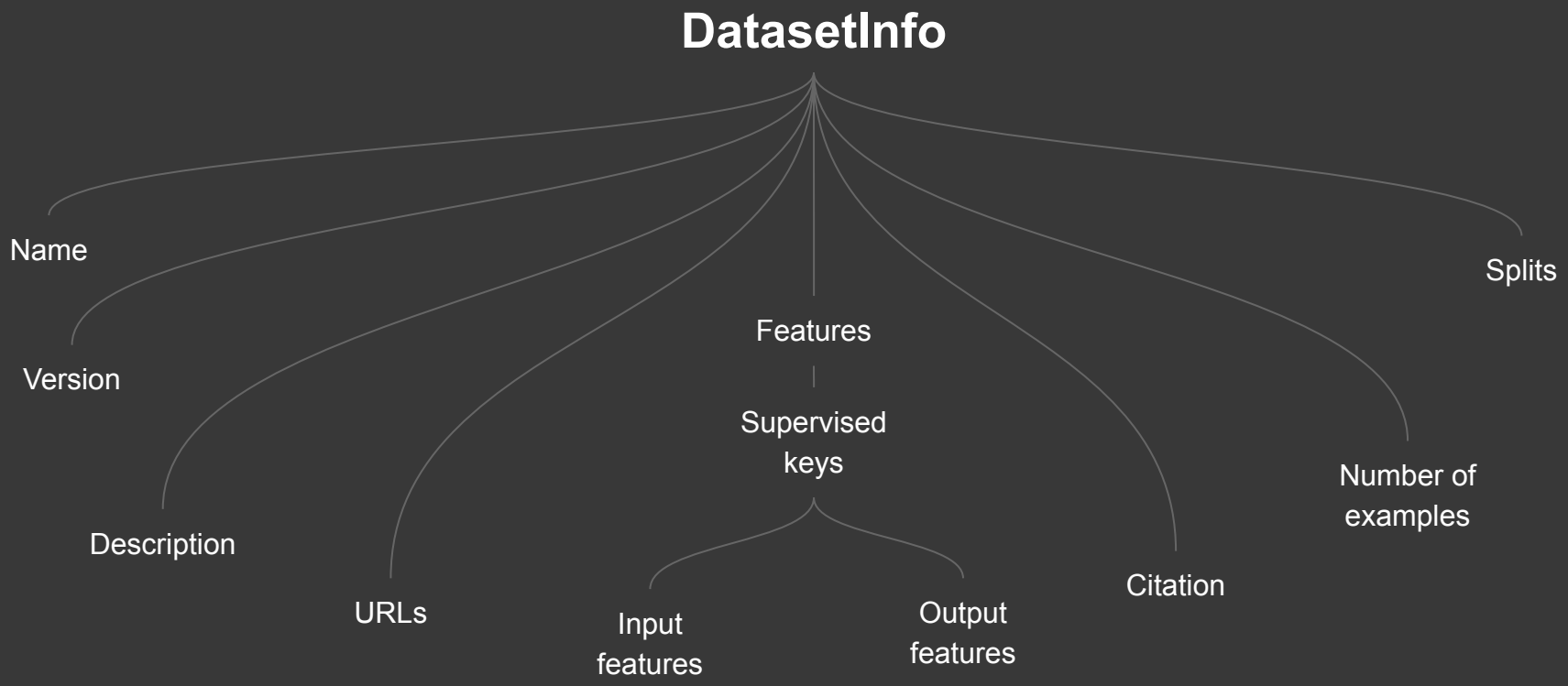
Choosing a dataset split

```
# 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}
```

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
```

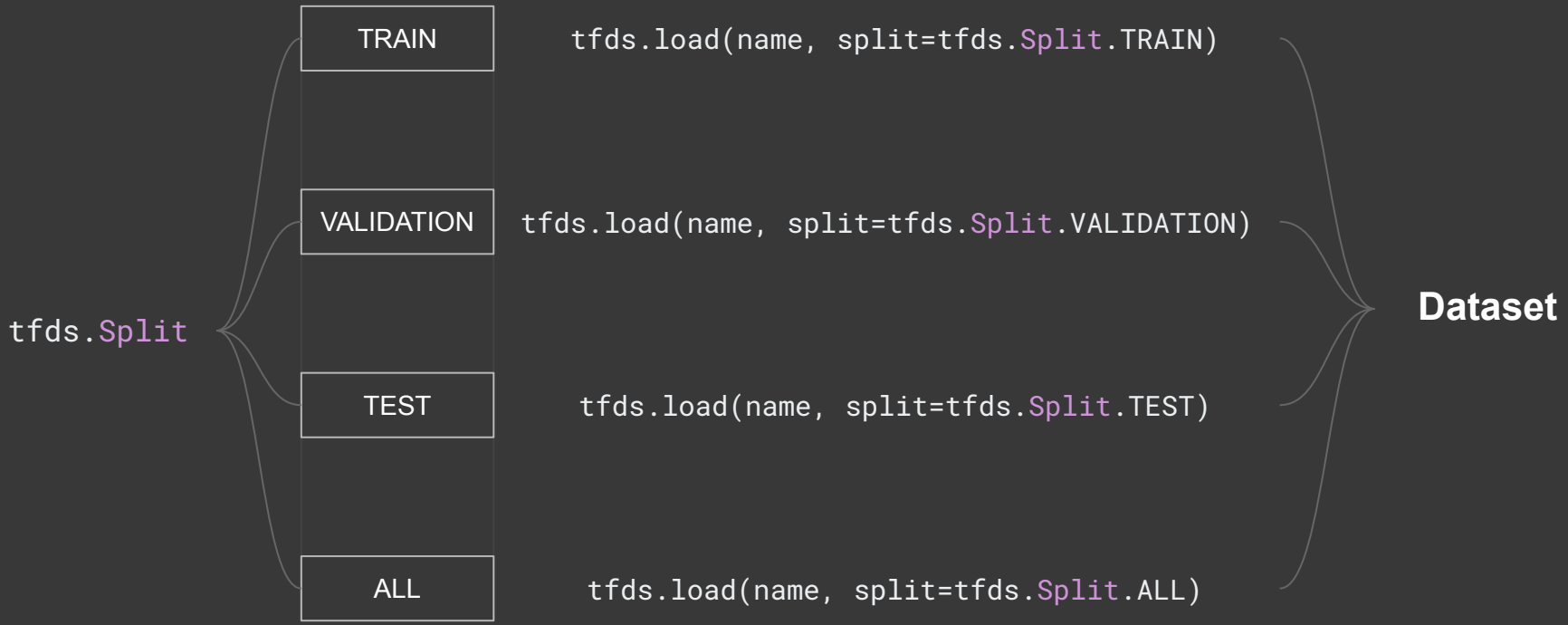
Loading a specific version

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

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,)
```

Using existing splits

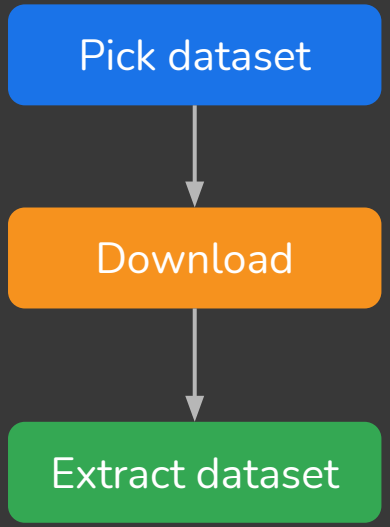


Non-conventional named splits

```
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)
```



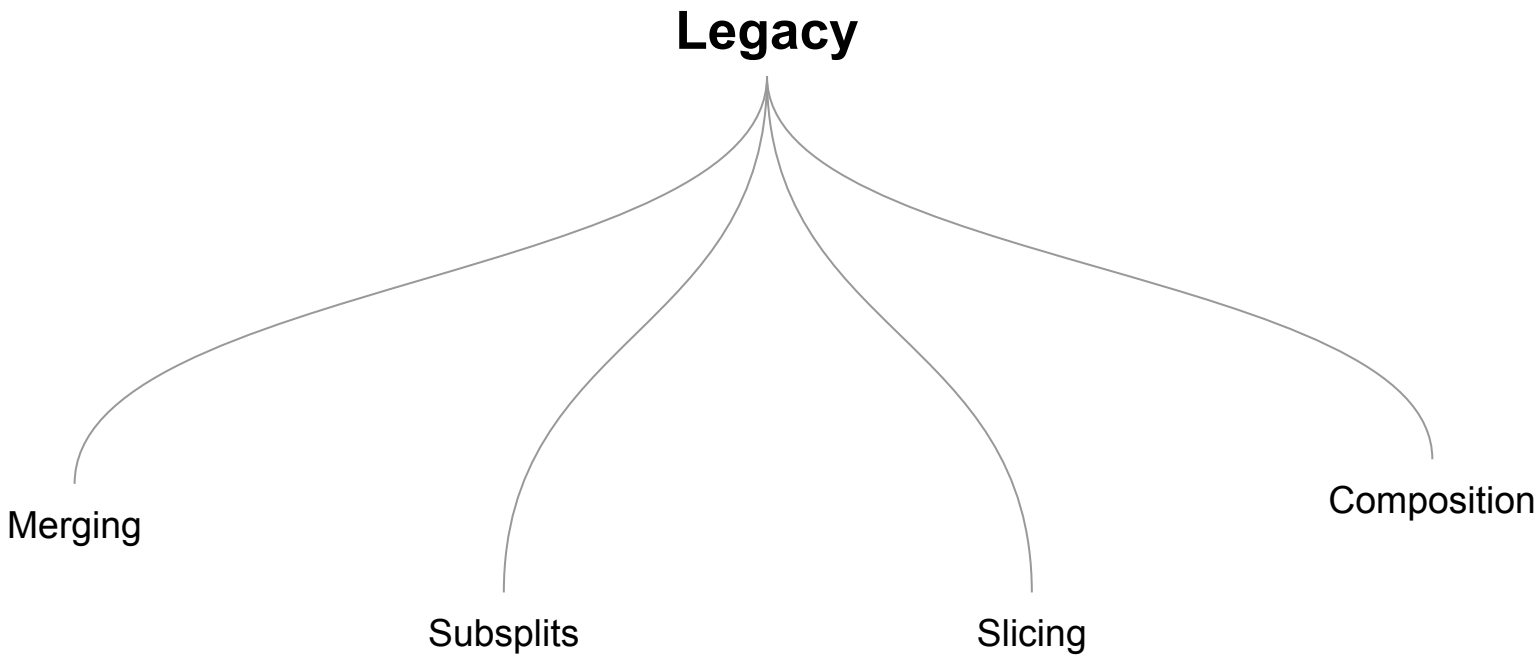
Two APIs

Legacy

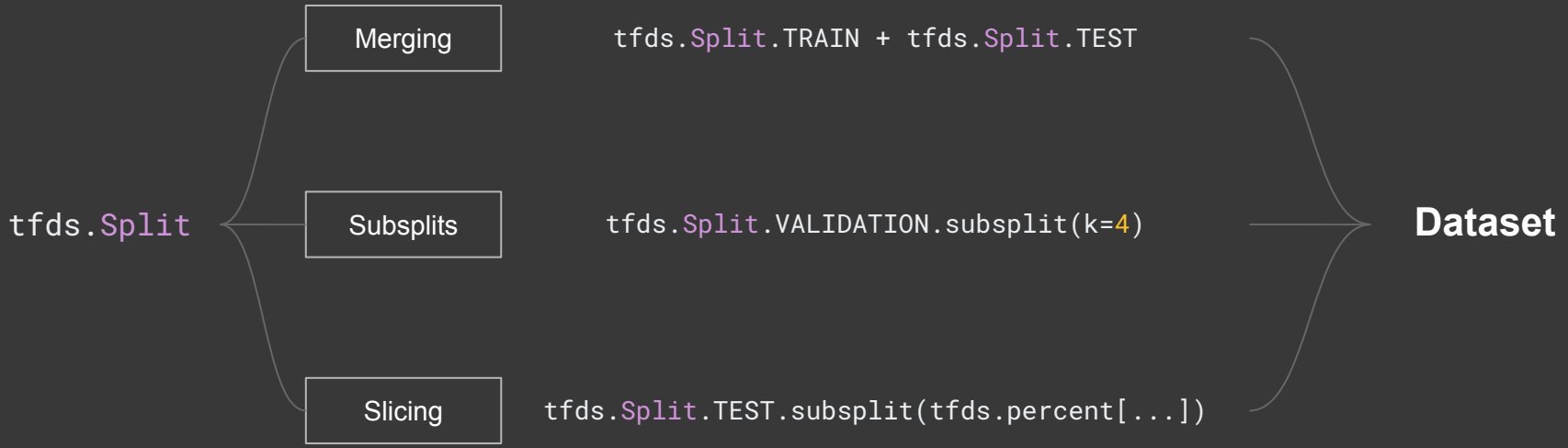
- Splitting with tfds.Split
- Support for all datasets in TFDS
- Features slicing

Splits API (new)

- Slicing with a string expression
- Newly added datasets support this


Legacy
Merging
Subsplits
Slicing
Composition

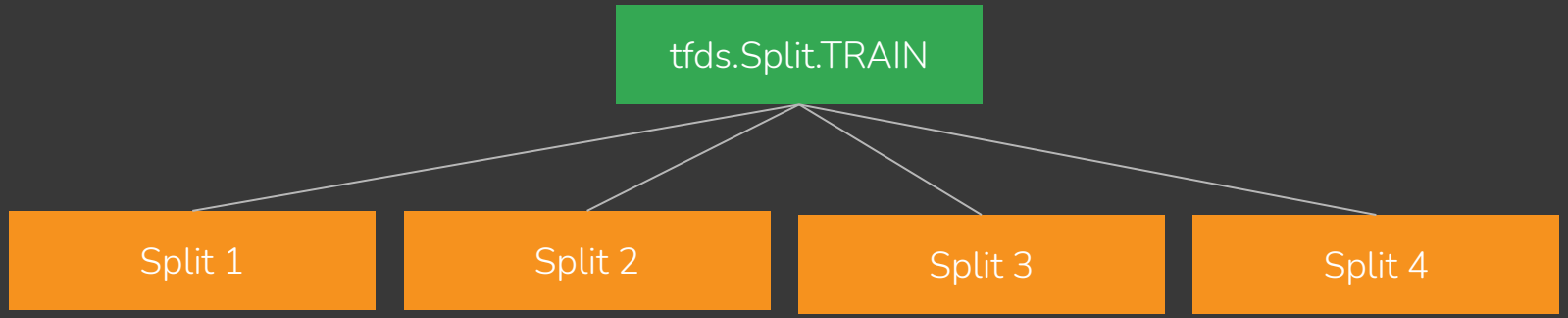
Legacy API


tfds.Split
Merging
tfds.Split.TRAIN + tfds.Split.TEST
Subsplits
tfds.Split.VALIDATION.subsplit(k=4)
Slicing
tfds.Split.TEST.subsplit(tfds.percent[...])
Dataset

Merging splits

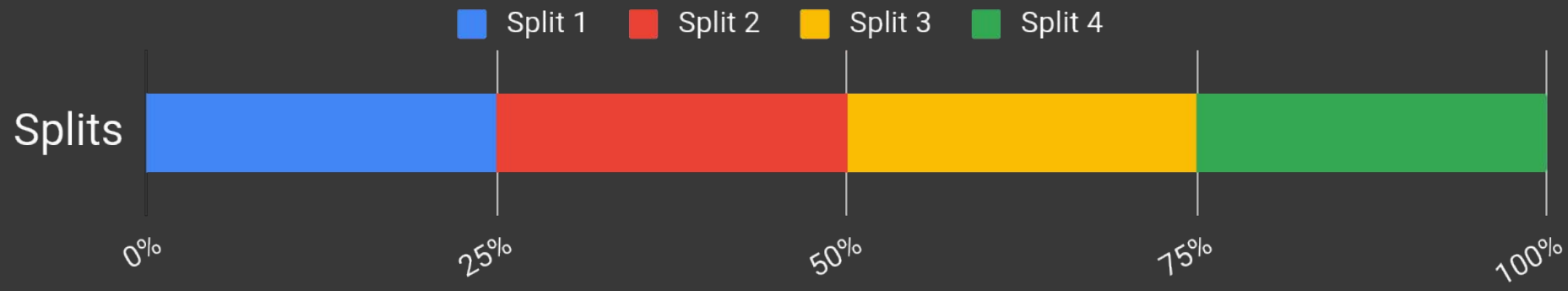

```
all = tfds.Split.TRAIN + tfds.Split.TEST
ds = tfds.load("mnist", split=all)

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

Creating subsplits



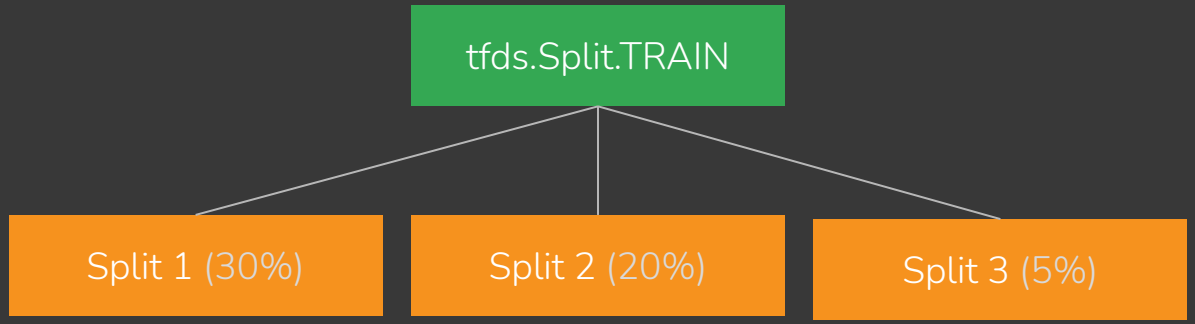

```
s1, s2, s3, s4 = tfds.Split.TRAIN.subsplit(k=4)

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

Creating subsplits



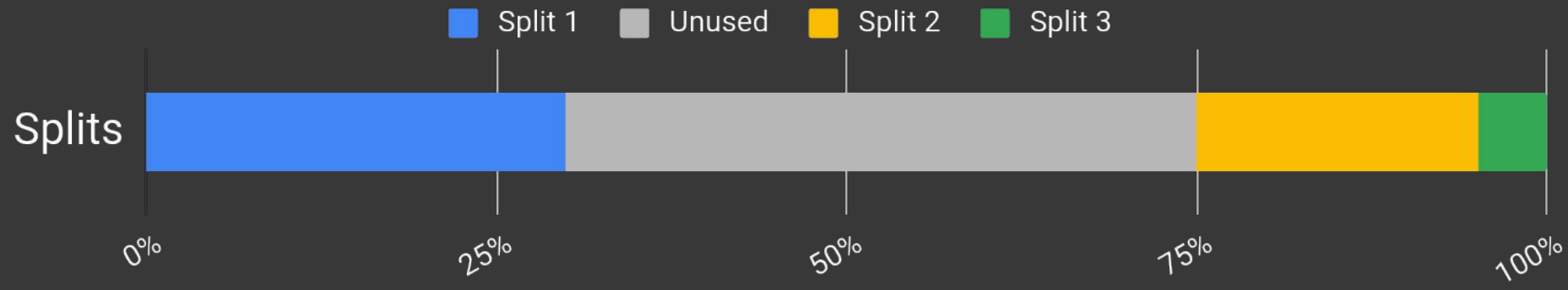

```
NamedSplit('train')(tfds.percent[0:25]) # Split 1
NamedSplit('train')(tfds.percent[25:50]) # Split 2
NamedSplit('train')(tfds.percent[50:75]) # Split 3
NamedSplit('train')(tfds.percent[75:100]) # Split 4
```

Percentage slicing



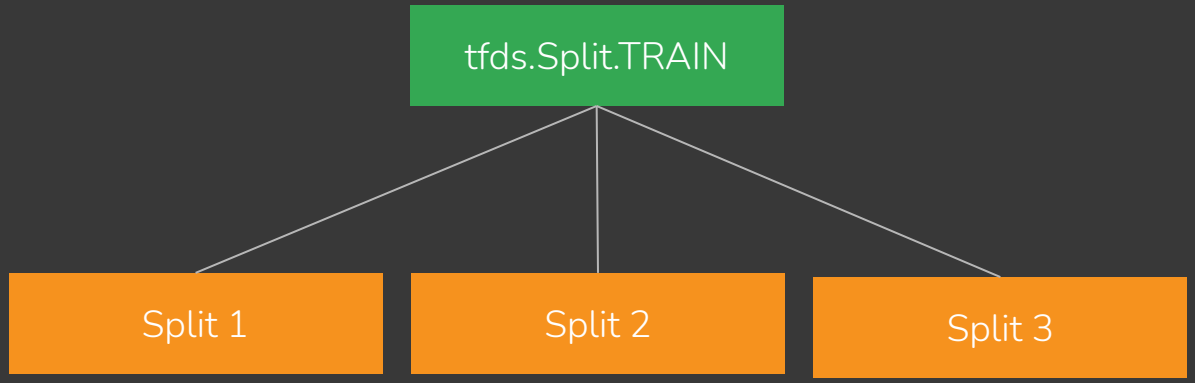

```
first_30_percent = tfds.Split.TRAIN.subsplit(tfds.percent[:30])
last_5_percent = tfds.Split.TRAIN.subsplit(tfds.percent[95:])
middle_20_percent = tfds.Split.TRAIN.subsplit(tfds.percent[75:95])
```

Percentage slicing



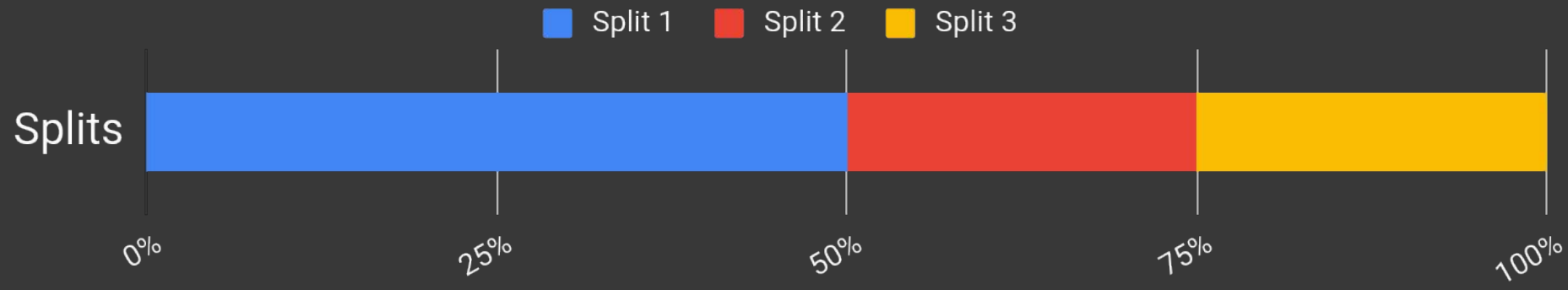

```
NamedSplit('train')(tfds.percent[:30]) # Split 1
NamedSplit('train')(tfds.percent[75:95]) # Split 2
NamedSplit('train')(tfds.percent[95:]) # Split 3
```

Weighted splits



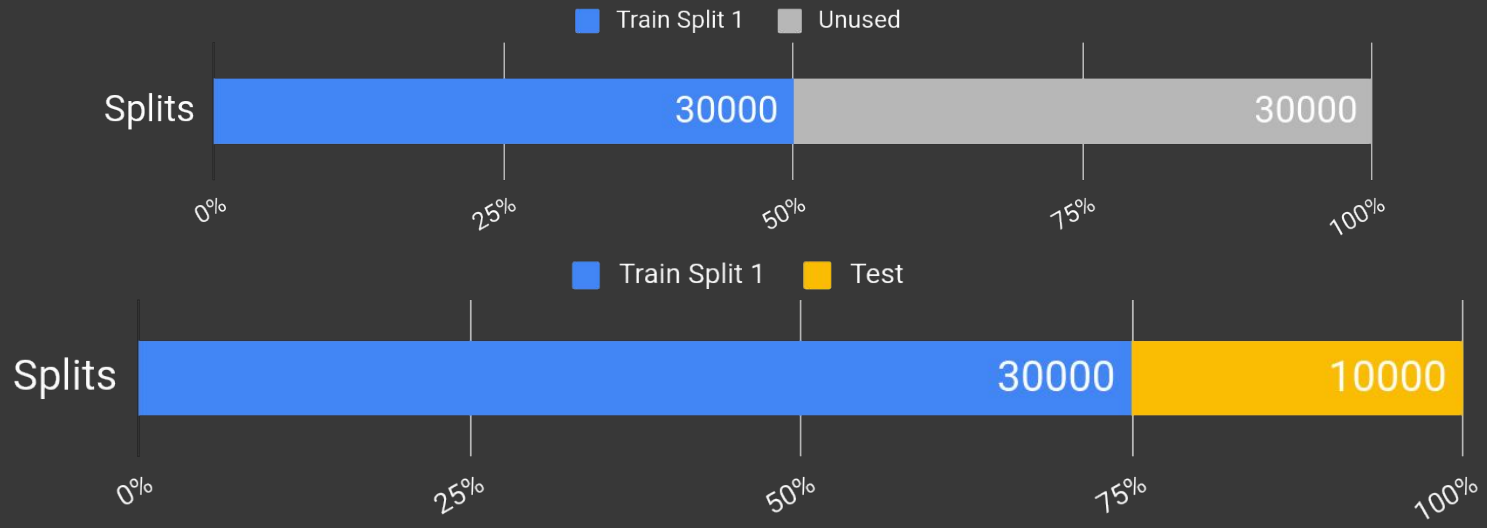

```
half, quarter1, quarter2 = tfds.Split.TRAIN.subsplit(weighted=[2, 1, 1])
```

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

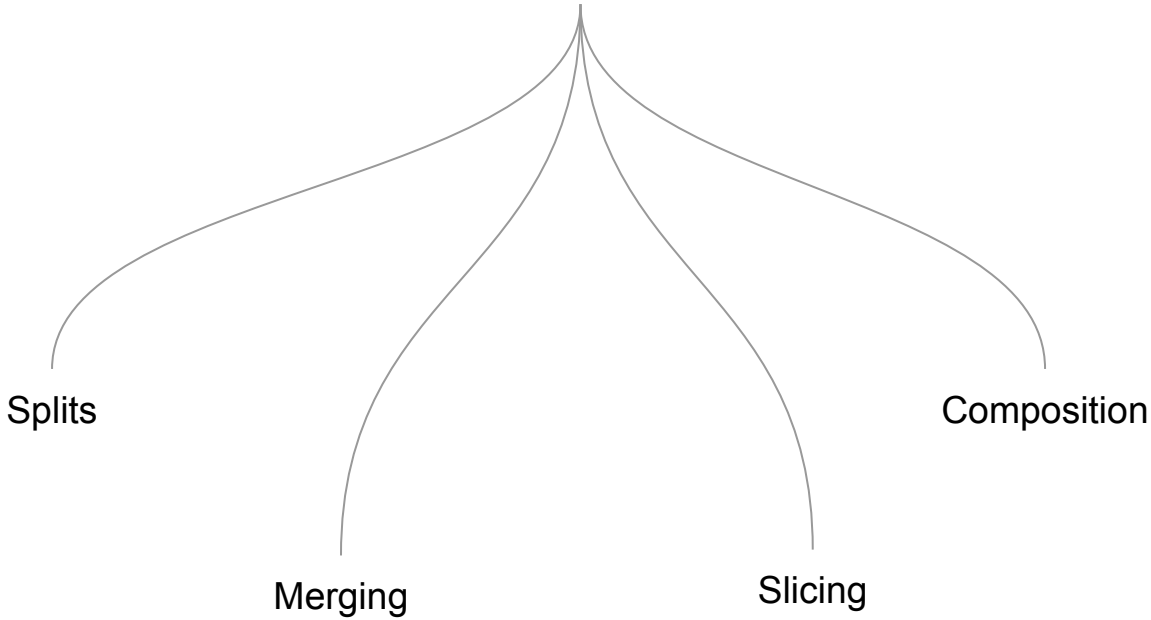



```
# Half of the TRAIN split plus the TEST split
first_50_train = tfds.Split.TRAIN.subsplit(tfds.percent[:50])
split = first_50_train + tfds.Split.TEST
dataset = tfds.load("mnist", split=split)
```

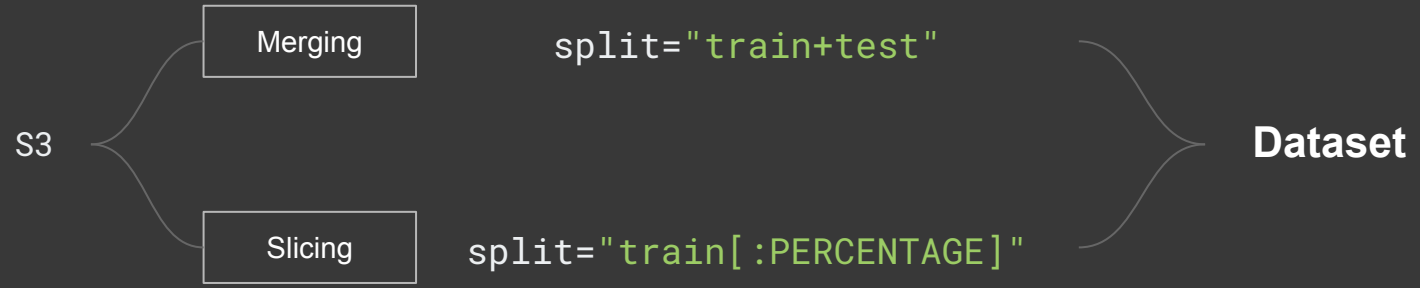
Invalid usages

```
# 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])
```

Splits API (S3)



Splits API


Merging
split="train+test"
S3
Dataset
Slicing
split="train[:PERCENTAGE]"

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

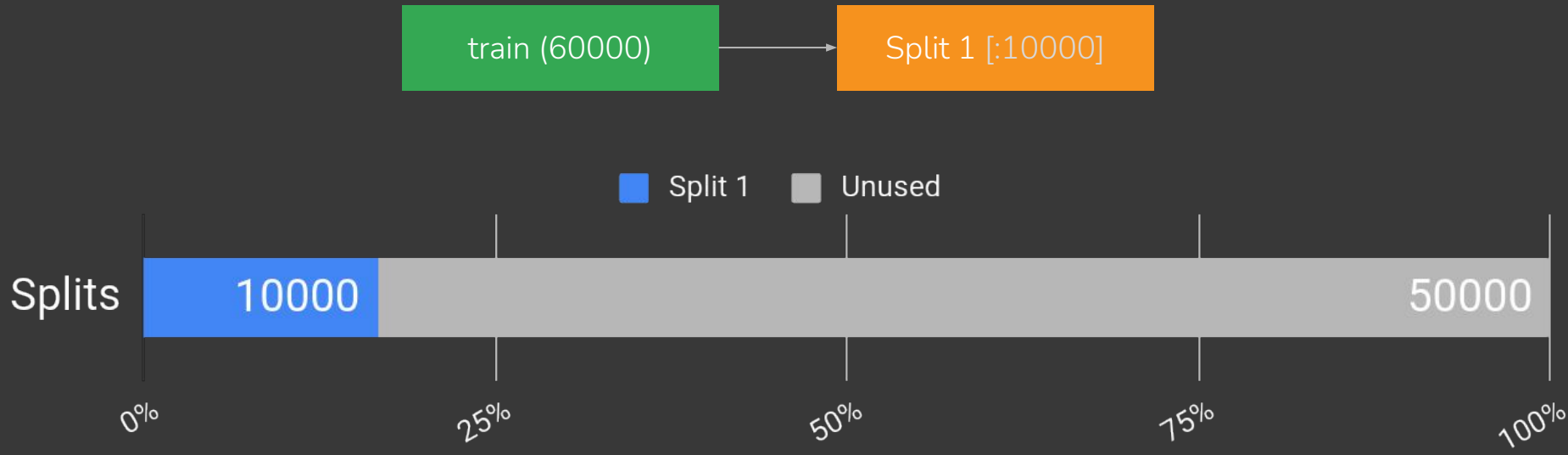
Distinct splits

```
# 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

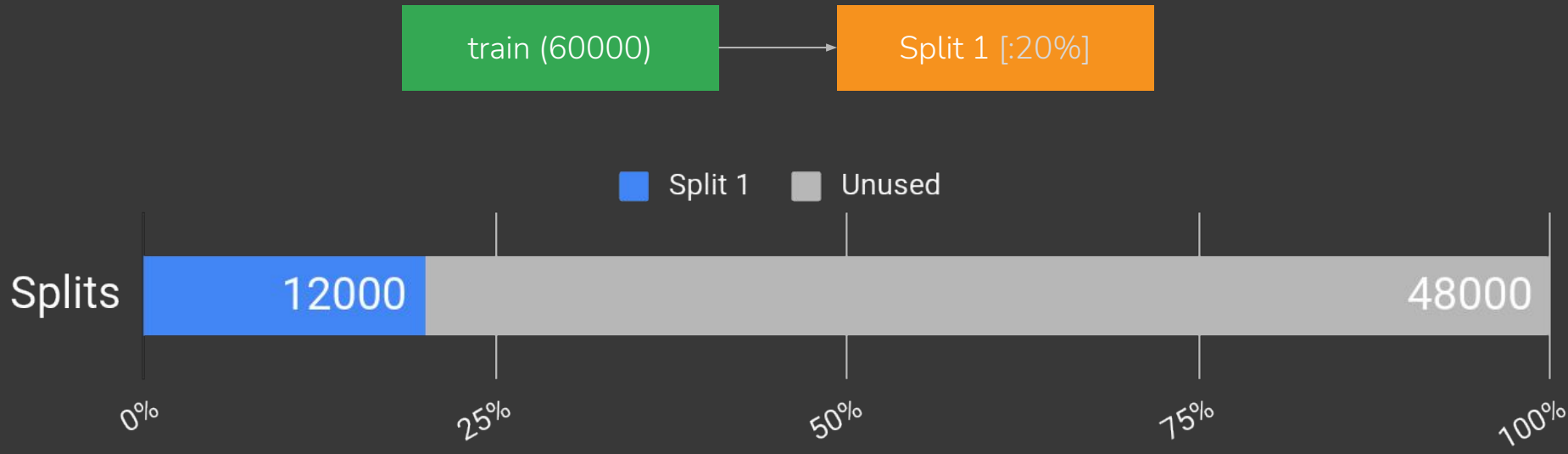

```
# The full `train` and `test` splits, concatenated together.
combined = tfds.load('mnist:3.*.*', split='train+test')
```

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%]`)

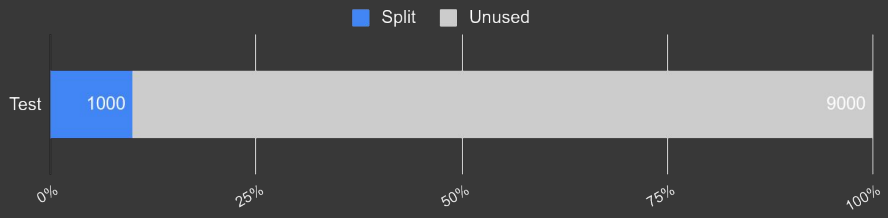
`[0%:20%] + [20%:100%]` (for a validation set of `[20%:40%]`)

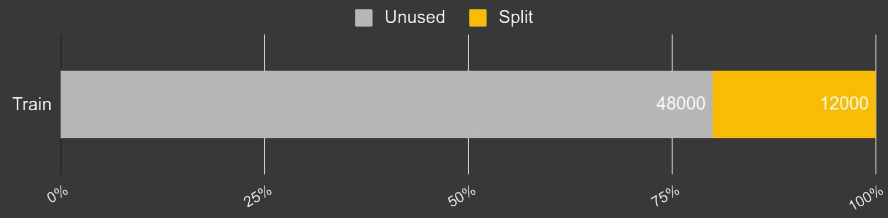
`[0%:80%]` (for a validation set of `[80%:100%]`)

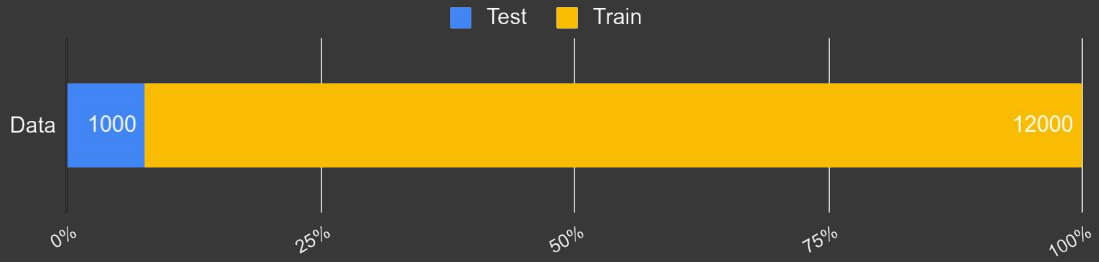
And so on ...


```
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%:]')
```