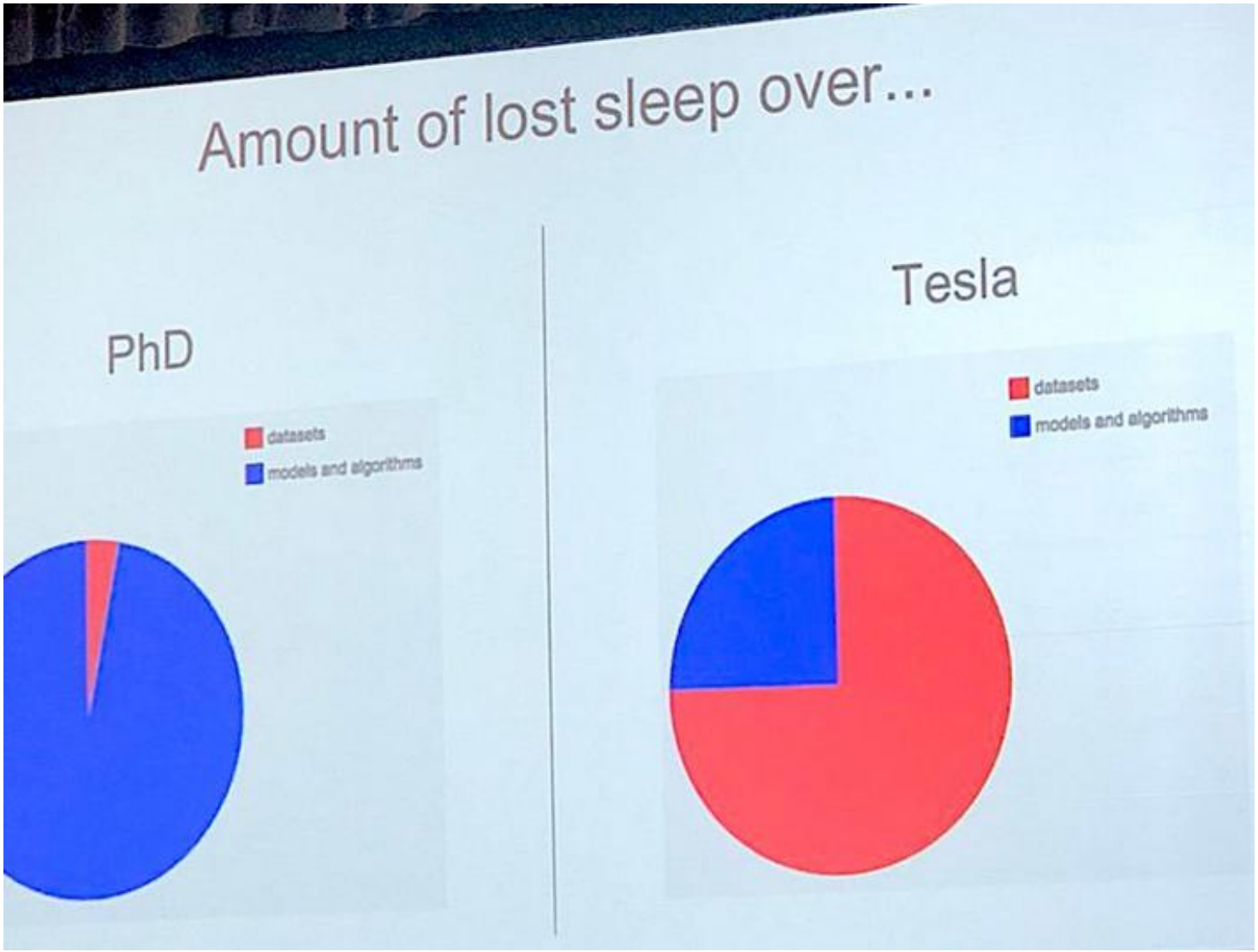
Data preprocessing with TensorFlow

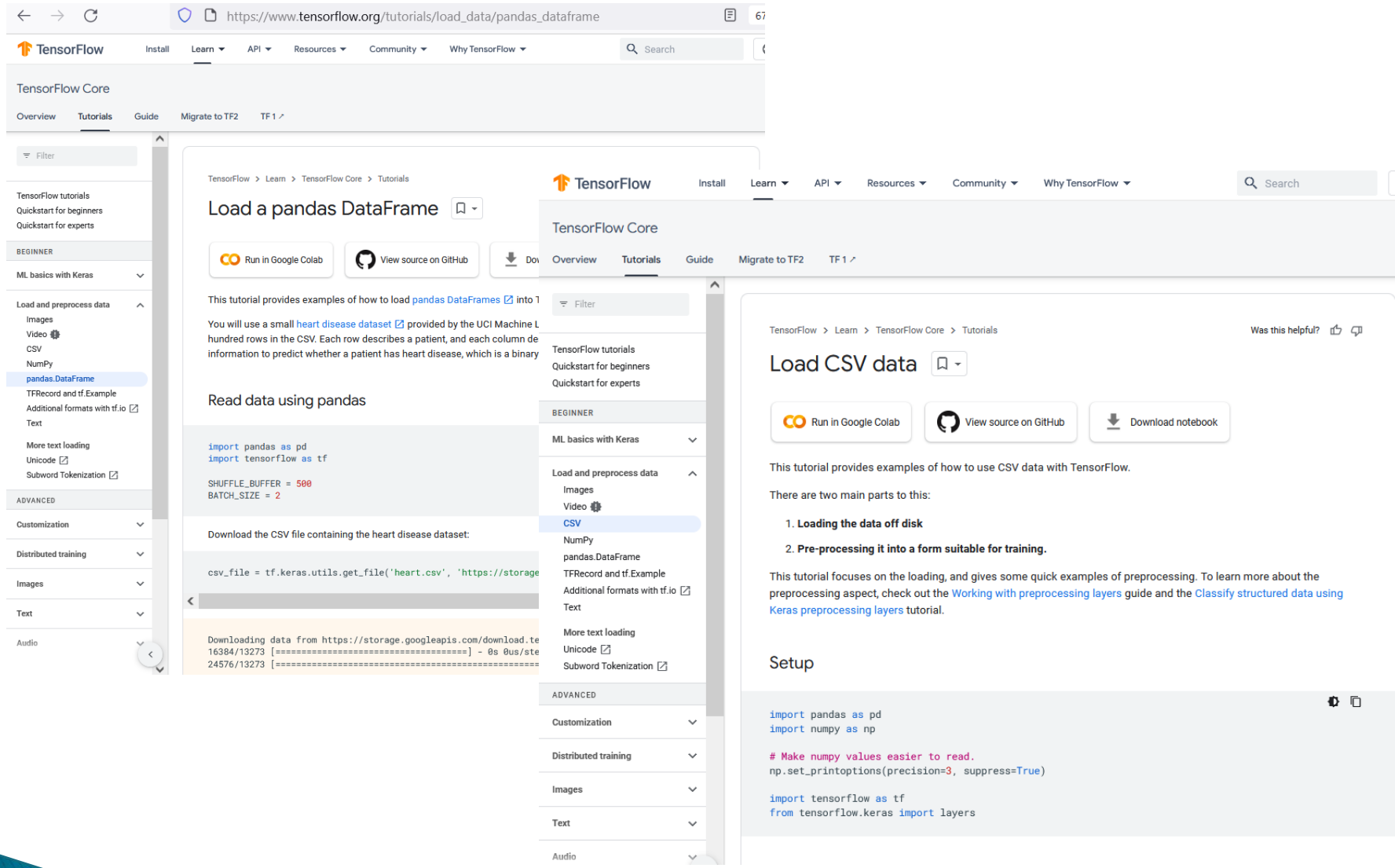
Data preprocessing



Andrej Karpathy showed this slide as part of his talk at Train AI

Photo by Lisha L

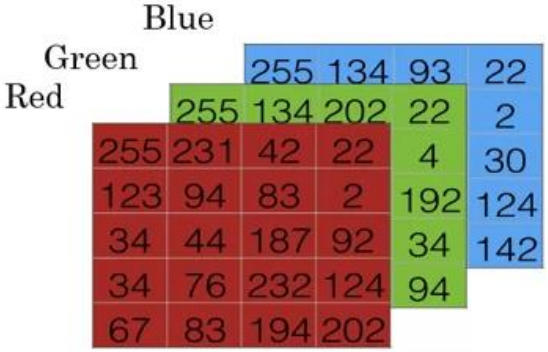
Load data



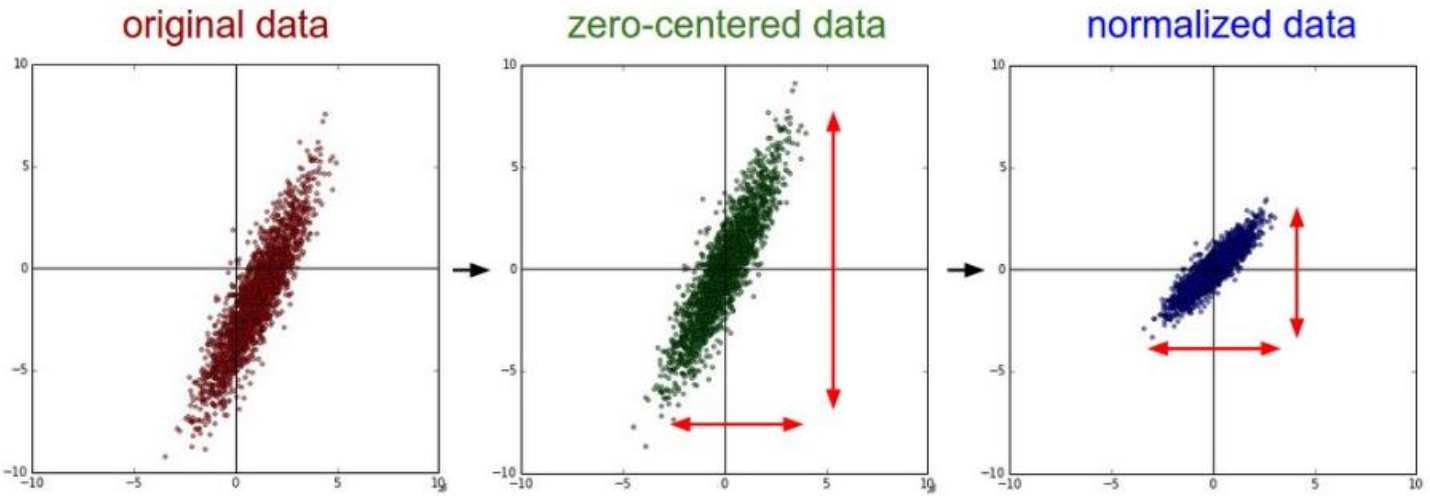
https://www.tensorflow.org/tutorials/load_data/pandas_dataframe

https://www.tensorflow.org/tutorials/load_data/csv

Normalization



$$x = \frac{x}{max}$$



$$x = \frac{x - \mu}{\sigma}$$

Filter

- MaxPool1D
- MaxPool2D
- MaxPool3D
- Maximum
- Minimum
- MultiHeadAttention
- Multiply
- Normalization
- PReLU
- Permute
- RNN
- RandomBrightness
- RandomContrast
- RandomCrop
- RandomFlip
- RandomHeight
- RandomRotation
- RandomTranslation
- RandomWidth
- RandomZoom
- ReLU
- RepeatVector
- Rescaling

tf.keras.layers.Normalization

Содержание

View source on GitHub

A preprocessing layer which normalizes continuous features.

Inherits From: `PreprocessingLayer`, `Layer`, `Module`

View aliases

```
tf.keras.layers.Normalization(
    axis=-1, mean=None, variance=None, invert=False, **kwargs
)
```

https://www.tensorflow.org/api_docs/python/tf/keras/layers/Normalization

```
tf.keras.layers.Normalization(
    axis=-1, mean=None, variance=None, invert=False, **kwargs
)
```

```
normalizer = tf.keras.layers.Normalization(axis=-1)
normalizer.adapt(numeric_features)
```

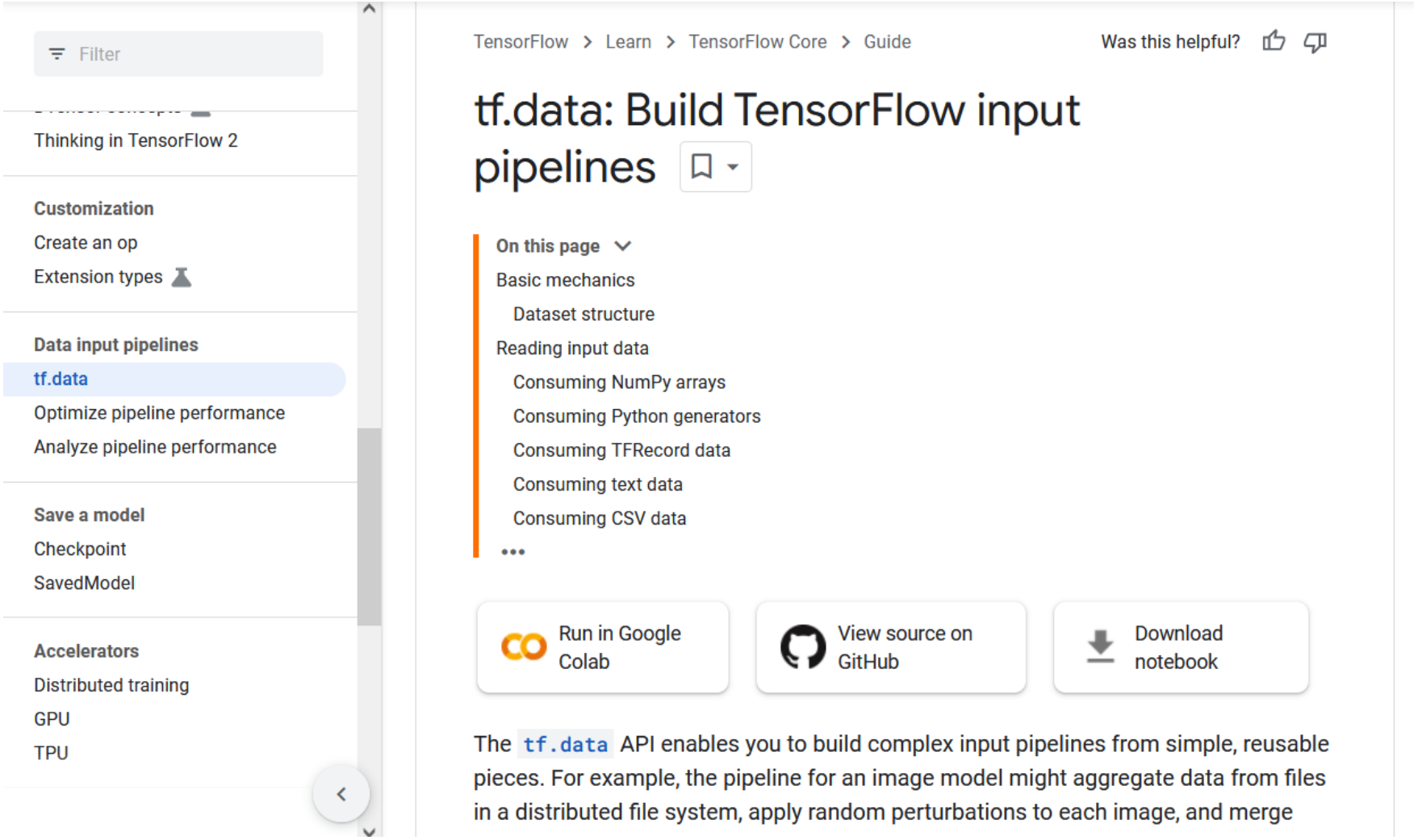
```
print(normalizer.mean.numpy())
print(normalizer.variance.numpy())
```

```
model = tf.keras.Sequential([
  normalizer,
  tf.keras.layers.Dense(10, activation='relu'),
  tf.keras.layers.Dense(10, activation='relu'),
  tf.keras.layers.Dense(1)
])
```

```
>>> adapt_data = np.array([1., 2., 3., 4., 5.], dtype='float32')
>>> input_data = np.array([1., 2., 3.], dtype='float32')
>>> layer = tf.keras.layers.Normalization(axis=None)
>>> layer.adapt(adapt_data)
>>> layer(input_data)
<tf.Tensor: shape=(3,), dtype=float32, numpy=
array([-1.4142135, -0.70710677, 0.], dtype=float32)>
```

```
>>> adapt_data = np.array([[0., 7., 4.],
...                        [2., 9., 6.],
...                        [0., 7., 4.],
...                        [2., 9., 6.]], dtype='float32')
>>> input_data = np.array([[0., 7., 4.]], dtype='float32')
>>> layer = tf.keras.layers.Normalization(axis=-1)
>>> layer.adapt(adapt_data)
>>> layer(input_data)
<tf.Tensor: shape=(1, 3), dtype=float32, numpy=
array([-1., -1., -1.], dtype=float32)>
```

```
>>> input_data = np.array([[1.], [2.], [3.]], dtype='float32')
>>> layer = tf.keras.layers.Normalization(mean=3., variance=2.)
>>> layer(input_data)
<tf.Tensor: shape=(3, 1), dtype=float32, numpy=
array([[-1.4142135 ],
       [-0.70710677],
       [ 0.        ]], dtype=float32)>
```



https://www.tensorflow.org/guide/data

tf.data.Dataset.from_tensor_slices

```
dataset = tf.data.Dataset.from_tensor_slices([8, 3, 0, 8, 2, 1])
dataset
```

```
for elem in dataset:
  print(elem.numpy())
```

```
8
3
0
8
2
1
```

tf.data.Dataset.from_tensor_slices

```
dataset1 = tf.data.Dataset.from_tensor_slices(
    tf.random.uniform([4, 10], minval=1, maxval=10, dtype=tf.int32))

dataset1
```

```
<TensorSliceDataset element_spec=TensorSpec(shape=(10,), dtype=tf.int32, name=
```

```
for z in dataset1:
  print(z.numpy())
```

```
[1 6 9 2 8 7 1 9 2 4]
[2 1 5 2 6 9 1 4 5 3]
[8 5 6 6 7 4 5 1 8 4]
[8 5 8 1 9 3 9 2 9 3]
```

Dataset.element_spec

```
dataset1 = tf.data.Dataset.from_tensor_slices(tf.random.uniform([4, 10]))

dataset1.element_spec
```

```
TensorSpec(shape=(10,), dtype=tf.float32, name=None)
```

```
dataset2 = tf.data.Dataset.from_tensor_slices(
   (tf.random.uniform([4]),
    tf.random.uniform([4, 100], maxval=100, dtype=tf.int32)))

dataset2.element_spec
```

```
(TensorSpec(shape=(), dtype=tf.float32, name=None),
 TensorSpec(shape=(100,), dtype=tf.int32, name=None))
```

```
dataset3 = tf.data.Dataset.zip((dataset1, dataset2))

dataset3.element_spec
```

```
(TensorSpec(shape=(10,), dtype=tf.float32, name=None),
 (TensorSpec(shape=(), dtype=tf.float32, name=None),
  TensorSpec(shape=(100,), dtype=tf.int32, name=None)))
```

Back to MNIST

```
train, test = tf.keras.datasets.mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
x_train, y_train = train
x_train = x_train/255

dataset_mnist = tf.data.Dataset.from_tensor_slices((x_train, y_train))
dataset_mnist
```

```
<TensorSliceDataset element_spec=(TensorSpec(shape=(28, 28), dtype=tf.float64, name=None),
TensorSpec(shape=(), dtype=tf.uint8, name=None))>
```

```
batched_dataset_mnist = dataset_mnist.batch(32)
```

```
>>> dataset = tf.data.Dataset.from_tensor_slices([1, 2, 3])
>>> dataset = dataset.map(lambda x: x*2)
>>> list(dataset.as_numpy_iterator())
[2, 4, 6]
```

```
>>> dataset = tf.data.Dataset.from_tensor_slices([1, 2, 3])
>>> dataset = dataset.filter(lambda x: x < 3)
>>> list(dataset.as_numpy_iterator())
[1, 2]
>>> # `tf.math.equal(x, y)` is required for equality comparison
>>> def filter_fn(x):
...   return tf.math.equal(x, 1)
>>> dataset = dataset.filter(filter_fn)
>>> list(dataset.as_numpy_iterator())
[1]
```

https://www.tensorflow.org/api_docs/python/tf/data/Dataset

Model.fit

data – dataframe, array

```
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

data – tf.data.dataset

Instead of passing `features` and `labels` to `Model.fit`, you pass the dataset:

```
model.fit(dataset_mnist, epochs=5)
```

```
batched_dataset = dataset.batch(4)
```

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

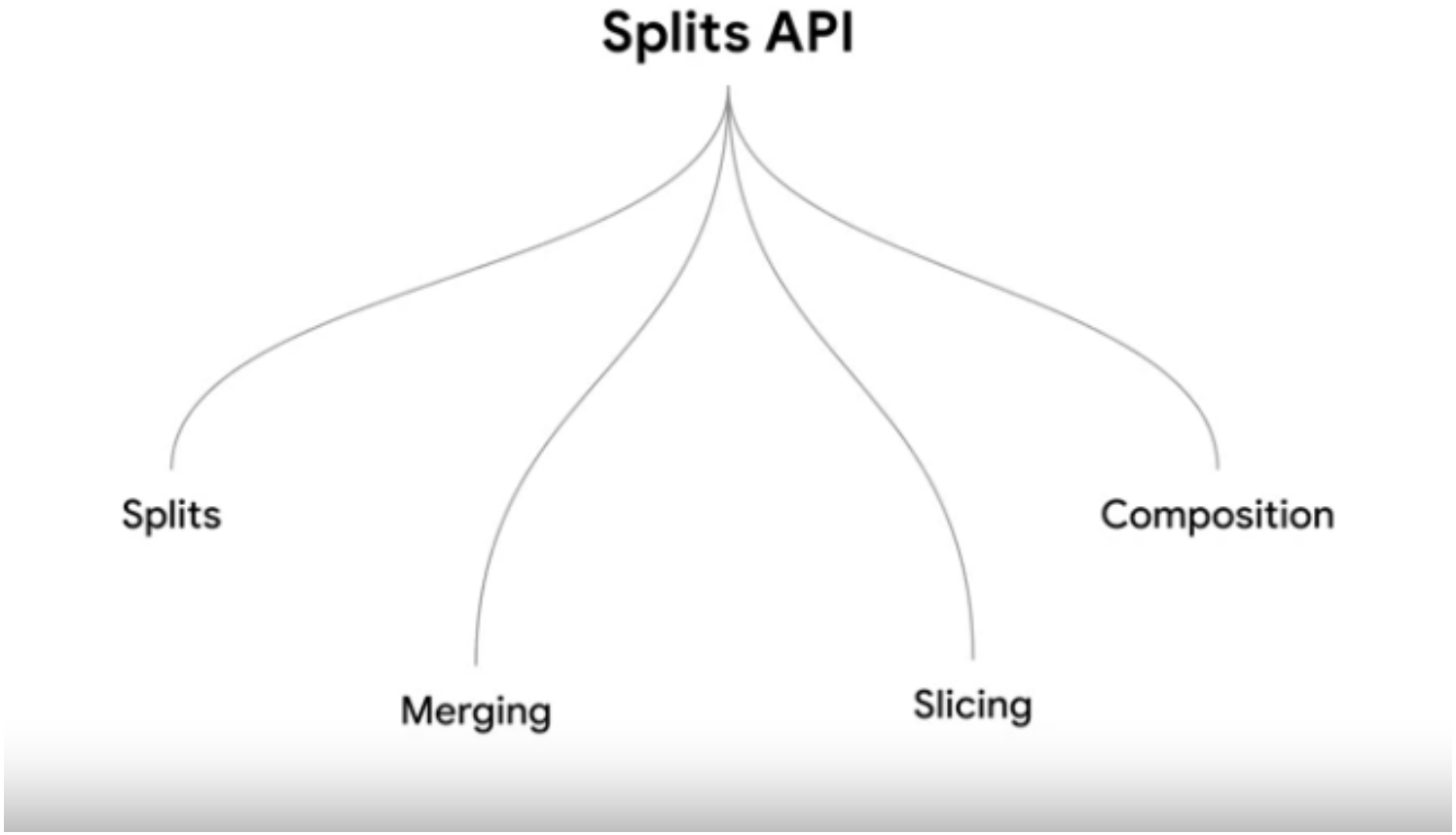
Text
IMDB reviews
Wikipedia
CNN - Daily Mail
SQuAD

Audio
NSynth
Groove

Video
UCF-101
Moving MNIST

Translate
WMT
TED multi-translate

https://www.coursera.org/learn/data-pipelines-tensorflow/home/info



https://www.coursera.org/learn/data-pipelines-tensorflow/home/info

Distinct splits

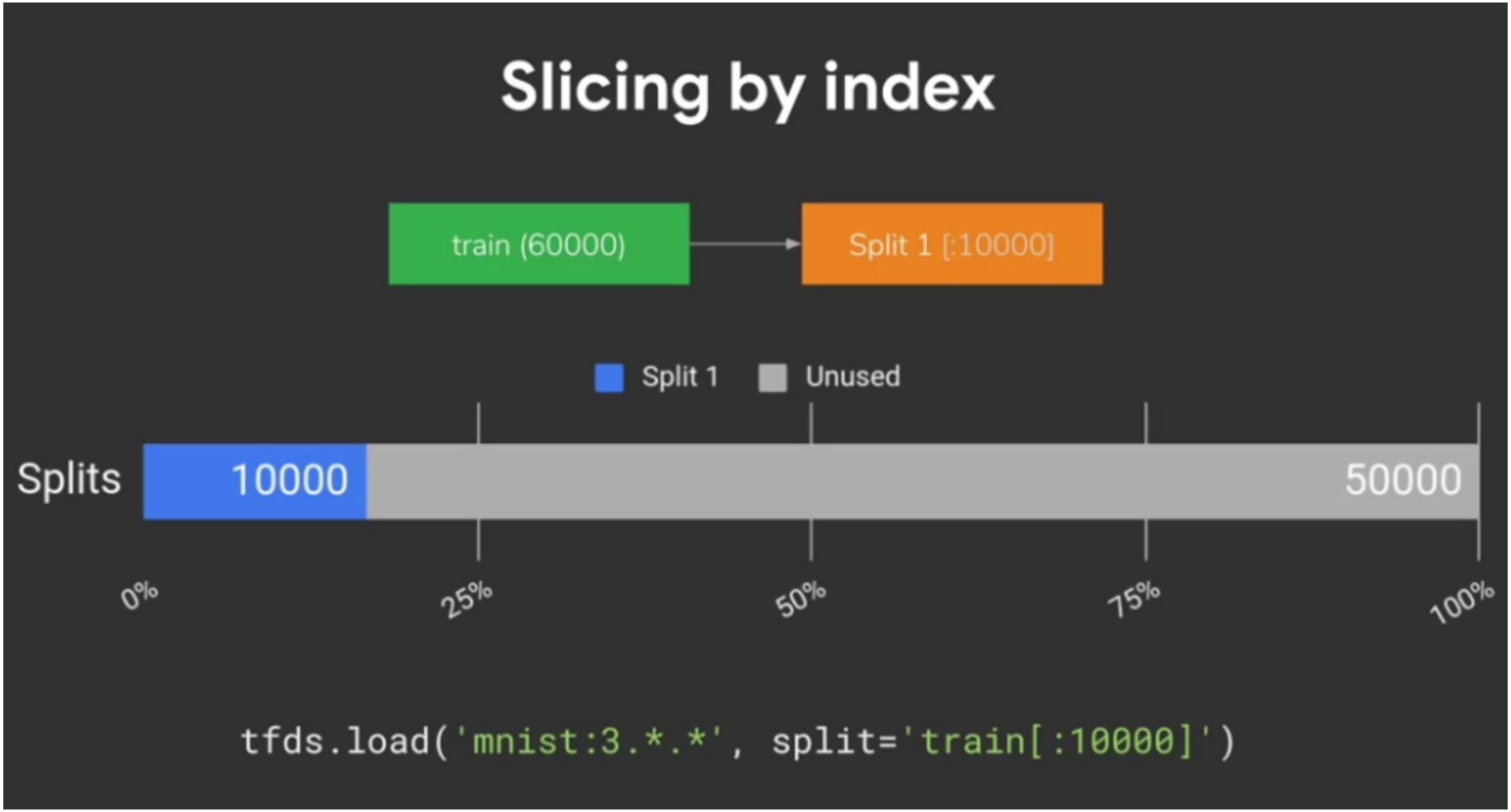
```
# The full `train` split and the full `test` split as two distinct datasets.
train_ds, test_ds = tfds.load('mnist:3.*.*', split=['train', 'test'])
```

https://www.coursera.org/learn/data-pipelines-tensorflow/home/info

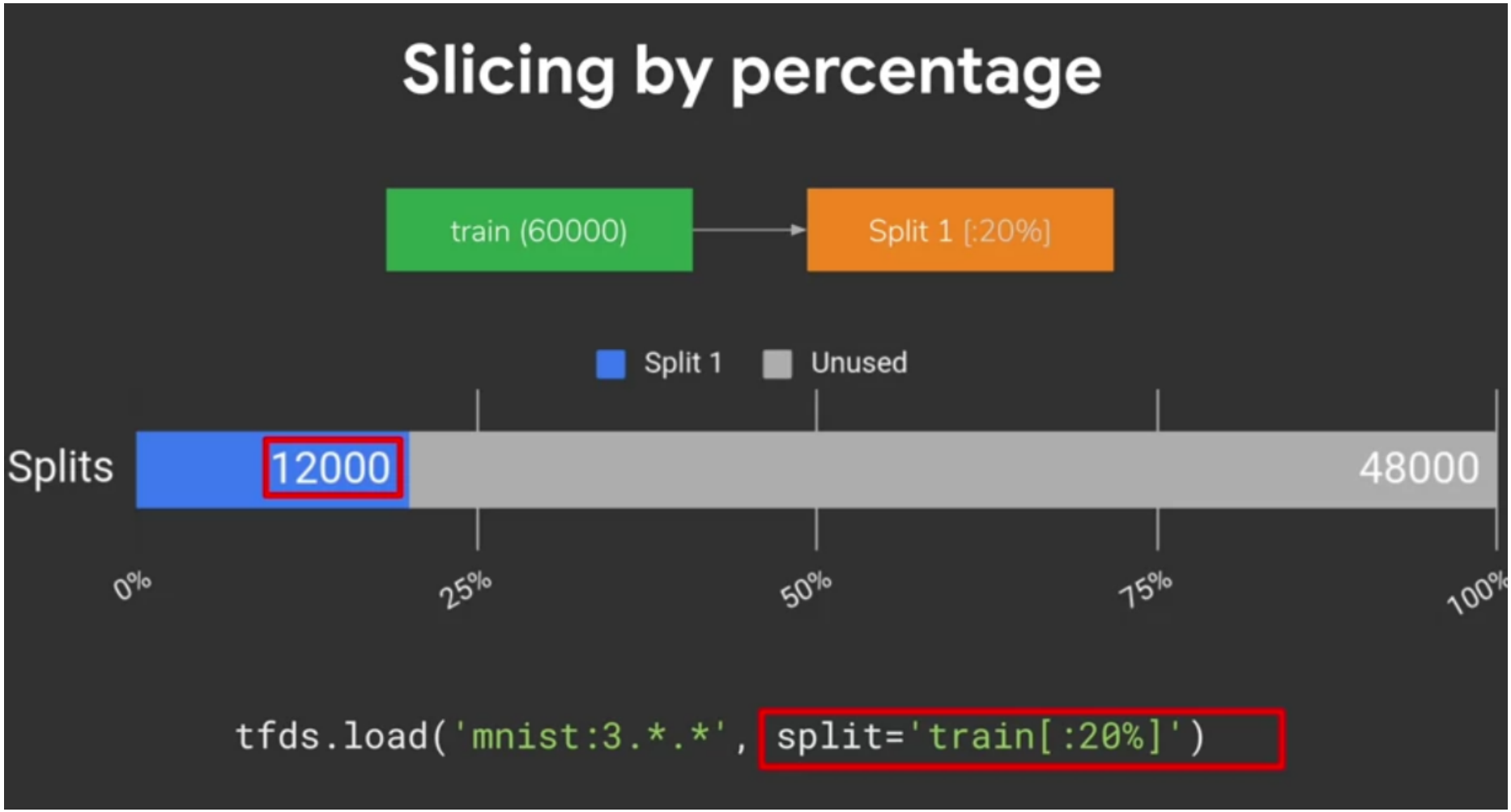
Merging


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

https://www.coursera.org/learn/data-pipelines-tensorflow/home/info

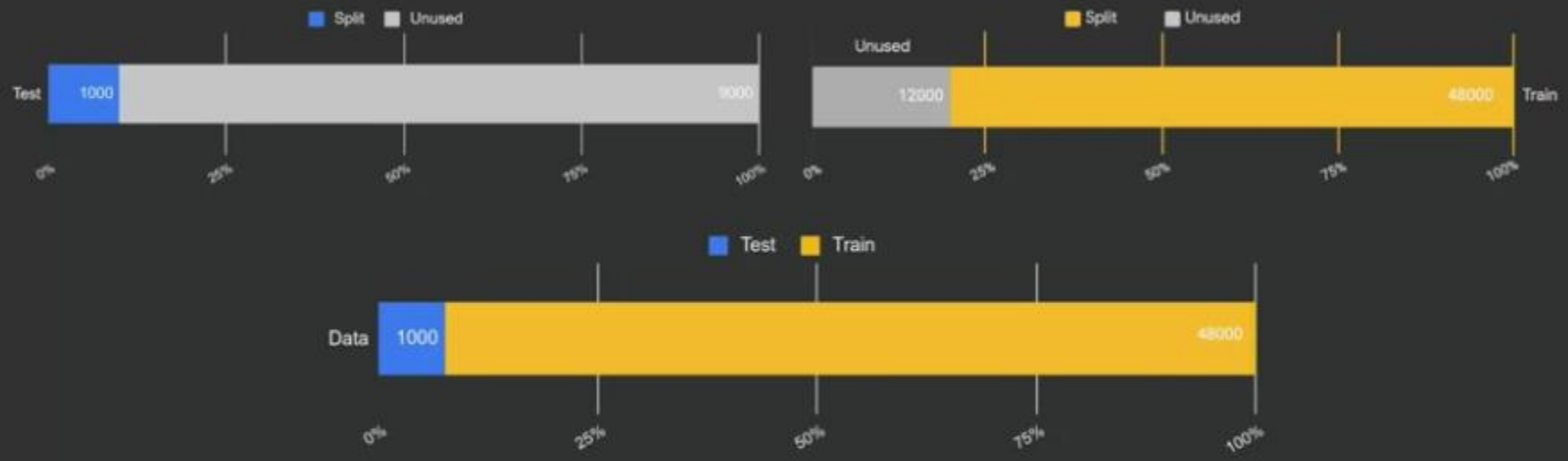


https://www.coursera.org/learn/data-pipelines-tensorflow/home/info



https://www.coursera.org/learn/data-pipelines-tensorflow/home/info

Composing operations




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

https://www.coursera.org/learn/data-pipelines-tensorflow/home/info