```
import tensorflow as tf
import tensorflow_datasets as tfds
```

Load a dataset

Load the MNIST dataset with the following arguments:

- `shuffle_files=True`: The MNIST data is only stored in a single file, but for larger datasets with multiple files on disk, it's good practice to shuffle them when training.
- `as_supervised=True`: Returns a tuple `(img, label)` instead of a dictionary `{'image': img, 'label': label}`.

```
(ds_train, ds_test), ds_info = tfds.load(
    'mnist',
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True,
)
```

```
WARNING:absl:Variant folder /root/tensorflow_datasets/mnist/3.0.1 has no dataset_info.json
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /root/t
Dl Completed...: 100%  4/4 [00:00<00:00,  4.96 url/s]
Dl Size...: 100%  10/10 [00:00<00:00, 12.77 MiB/s]
Extraction completed...: 100%  4/4 [00:00<00:00,  5.27 file/s]
Dataset mnist downloaded and prepared to /root/tensorflow_datasets/mnist/3.0.1. Subsequent calls will reuse this data.
```

```
def normalize_img(image, label):
  """Normalizes images: `uint8` -> `float32`."""
  return tf.cast(image, tf.float32) / 255., label

ds_train = ds_train.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_train = ds_train.cache()
ds_train = ds_train.shuffle(ds_info.splits['train'].num_examples)
ds_train = ds_train.batch(128)
ds_train = ds_train.prefetch(tf.data.AUTOTUNE)
```

Build an evaluation pipeline

Your testing pipeline is similar to the training pipeline with small differences:

- You don't need to call `tf.data.Dataset.shuffle`.
- Caching is done after batching because batches can be the same between epochs.

```
ds_test = ds_test.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_test = ds_test.batch(128)
ds_test = ds_test.cache()
ds_test = ds_test.prefetch(tf.data.AUTOTUNE)
```

Step 2: Create and train the model

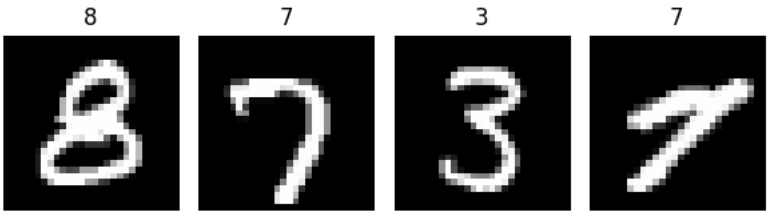
Plug the TFDS input pipeline into a simple Keras model, compile the model, and train it.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10)
])
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
```

```
)

model.fit(
    ds_train,
    epochs=6,
    validation_data=ds_test,
)
```

```
Epoch 1/6
/usr/local/lib/python3.12/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`inp
  super().__init__(**kwargs)
469/469 ━━━━━━━━━━━━━━━━━━━━ 12s 7ms/step - loss: 0.6021 - sparse_categorical_accuracy: 0.8372 - val_loss: 0.2079 - val_sparse_c
Epoch 2/6
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.1845 - sparse_categorical_accuracy: 0.9485 - val_loss: 0.1390 - val_sparse_ca
Epoch 3/6
469/469 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - loss: 0.1253 - sparse_categorical_accuracy: 0.9638 - val_loss: 0.1188 - val_sparse_ca
Epoch 4/6
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.0935 - sparse_categorical_accuracy: 0.9736 - val_loss: 0.0942 - val_sparse_ca
Epoch 5/6
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - loss: 0.0753 - sparse_categorical_accuracy: 0.9793 - val_loss: 0.0946 - val_sparse_ca
Epoch 6/6
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.0619 - sparse_categorical_accuracy: 0.9822 - val_loss: 0.0847 - val_sparse_ca
<keras.src.callbacks.history.History at 0x7a83d3200ad0>
```

```
plt.figure(figsize=(6,2))  # overall size (adjust if needed)

for i, (image, label) in enumerate(ds_train.unbatch().take(4)):
    plt.subplot(1, 4, i+1)  # 1 row, 4 columns
    plt.imshow(image.numpy().squeeze(), cmap='gray')
    plt.title(label.numpy())
    plt.axis('off')

plt.tight_layout()
plt.show()
```




```
Start coding or generate with AI.
```