TensorFlow 2 quickstart for experts

Run in Google (https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/quic Colab

This is a Google Colaboratory (https://colab.research.google.com/notebooks/welcome.ipynb) notebook file. Python programs are run directly in the browser—a great way to learn and use TensorFlow. To follow this tutorial, run the notebook in Google Colab by clicking the button at the top of this page.

1. In Colab, connect to a Python runtime: At the top-right of the menu bar, select *CONNECT*.
2. Run all the notebook code cells: Select *Runtime* > *Run all*.

Download and install TensorFlow 2. Import TensorFlow into your program:

Jpgrade **pip** to install the TensorFlow 2 package. See the install guide (https://www.tensorflow.org/instal
:.

Import TensorFlow into your program:

```
't tensorflow as tf

 tensorflow.keras.layers import Dense, Flatten, Conv2D
 tensorflow.keras import Model
```

Load and prepare the MNIST dataset (http://yann.lecun.com/exdb/mnist/).

```
: = tf.keras.datasets.mnist

'ain, y_train), (x_test, y_test) = mnist.load_data()
iin, x_test = x_train / 255.0, x_test / 255.0

l a channels dimension
iin = x_train[..., tf.newaxis]
;t = x_test[..., tf.newaxis]
```

Use **`tf.data`** (/api_docs/python/tf/data) to batch and shuffle the dataset:

```
_ds = tf.data.Dataset.from_tensor_slices(
x_train, y_train)).shuffle(10000).batch(32)

ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)
```

Build the **`tf.keras`** (/api_docs/python/tf/keras) model using the Keras model subclassing API (https://www.tensorflow.org/guide/keras#model_subclassing):

```
 MyModel(Model):
 __init__(self):
uper(MyModel, self).__init__()
elf.conv1 = Conv2D(32, 3, activation='relu')
elf.flatten = Flatten()
elf.d1 = Dense(128, activation='relu')
elf.d2 = Dense(10)

 call(self, x):
 = self.conv1(x)
 = self.flatten(x)
 = self.d1(x)
eturn self.d2(x)

ate an instance of the model
 = MyModel()
```

Choose an optimizer and loss function for training:

```
object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)

izer = tf.keras.optimizers.Adam()
```

Select metrics to measure the loss and the accuracy of the model. These metrics accumulate the values over epochs and then print the overall result.


```
_loss = tf.keras.metrics.Mean(name='train_loss')
_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')
```

```
.loss = tf.keras.metrics.Mean(name='test_loss')
.accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='test_accuracy')
```

Use **`tf.GradientTape`** (/api_docs/python/tf/GradientTape) to train the model:

```
'unction
:rain_step(images, labels):
:h tf.GradientTape() as tape:
! training=True is only needed if there are layers with different
! behavior during training versus inference (e.g. Dropout).
)redictions = model(images, training=True)
.oss = loss_object(labels, predictions)
ıdients = tape.gradient(loss, model.trainable_variables)
:imizer.apply_gradients(zip(gradients, model.trainable_variables))

ıin_loss(loss)
ıin_accuracy(labels, predictions)
```

Test the model:

```
'unction
:est_step(images, labels):
:raining=False is only needed if there are layers with different
)ehavior during training versus inference (e.g. Dropout).
:dictions = model(images, training=False)
.oss = loss_object(labels, predictions)

;t_loss(t_loss)
;t_accuracy(labels, predictions)
```

```
IS = 5

:poch in range(EPOCHS):
?eset the metrics at the start of the next epoch
ıin_loss.reset_states()
ıin_accuracy.reset_states()
;t_loss.reset_states()
;t_accuracy.reset_states()

' images, labels in train_ds:
:rain_step(images, labels)
```

```
 test_images, test_labels in test_ds:
:est_step(test_images, test_labels)

ıplate = 'Epoch {}, Loss: {}, Accuracy: {}, Test Loss: {}, Test Accuracy: {}'
.nt(template.format(epoch + 1,
                    train_loss.result(),
                    train_accuracy.result() * 100,
                    test_loss.result(),
                    test_accuracy.result() * 100))
```

```
:NG:tensorflow:Layer my_model is casting an input tensor from dtype float64 to

ıu intended to run this layer in float32, you can safely ignore this warning.

ıange all layers to have dtype float64 by default, call `tf.keras.backend.set_

ı 1, Loss: 0.13576626777648926, Accuracy: 95.89666748046875, Test Loss: 0.0658
ı 2, Loss: 0.04346510395407677, Accuracy: 98.66667175292969, Test Loss: 0.0584
ı 3, Loss: 0.024708667770028114, Accuracy: 99.1883316040039, Test Loss: 0.0513
ı 4, Loss: 0.014630611054599285, Accuracy: 99.49500274658203, Test Loss: 0.056
ı 5, Loss: 0.010477297939360142, Accuracy: 99.61333465576172, Test Loss: 0.064
```

The image classifier is now trained to ~98% accuracy on this dataset. To learn more, read the TensorFlow tutorials (https://www.tensorflow.org/tutorials).

Last updated 2020-06-12.