C1_W1_Lab_1_functional-practice

February 8, 2021

1 Ungraded Lab: Practice with the Keras Functional API

This lab will demonstrate how to build models with the Functional syntax. You'll build one using the Sequential API and see how you can do the same with the Functional API. Both will arrive at the same architecture and you can train and evaluate it as usual.

1.1 Imports

[1]:
```
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass

import tensorflow as tf
from tensorflow.python.keras.utils.vis_utils import plot_model
import pydot
from tensorflow.keras.models import Model
```

1.2 Sequential API

Here is how we use the `Sequential()` class to build a model.

[2]:
```
def build_model_with_sequential():

    # instantiate a Sequential class and linearly stack the layers of your model
    seq_model = tf.keras.models.Sequential([tf.keras.layers.
 ↪Flatten(input_shape=(28, 28)),
                                            tf.keras.layers.Dense(128,␣
 ↪activation=tf.nn.relu),
                                            tf.keras.layers.Dense(10,␣
 ↪activation=tf.nn.softmax)])
    return seq_model
```

1.3 Functional API

And here is how you build the same model above with the functional syntax.

[3]:

```
def build_model_with_functional():

    # instantiate the input Tensor
    input_layer = tf.keras.Input(shape=(28, 28))

    # stack the layers using the syntax: new_layer()(previous_layer)
    flatten_layer = tf.keras.layers.Flatten()(input_layer)
    first_dense = tf.keras.layers.Dense(128, activation=tf.nn.
 ↪relu)(flatten_layer)
    output_layer = tf.keras.layers.Dense(10, activation=tf.nn.
 ↪softmax)(first_dense)

    # declare inputs and outputs
    func_model = Model(inputs=input_layer, outputs=output_layer)

    return func_model
```

1.4 Build the model and visualize the model graph

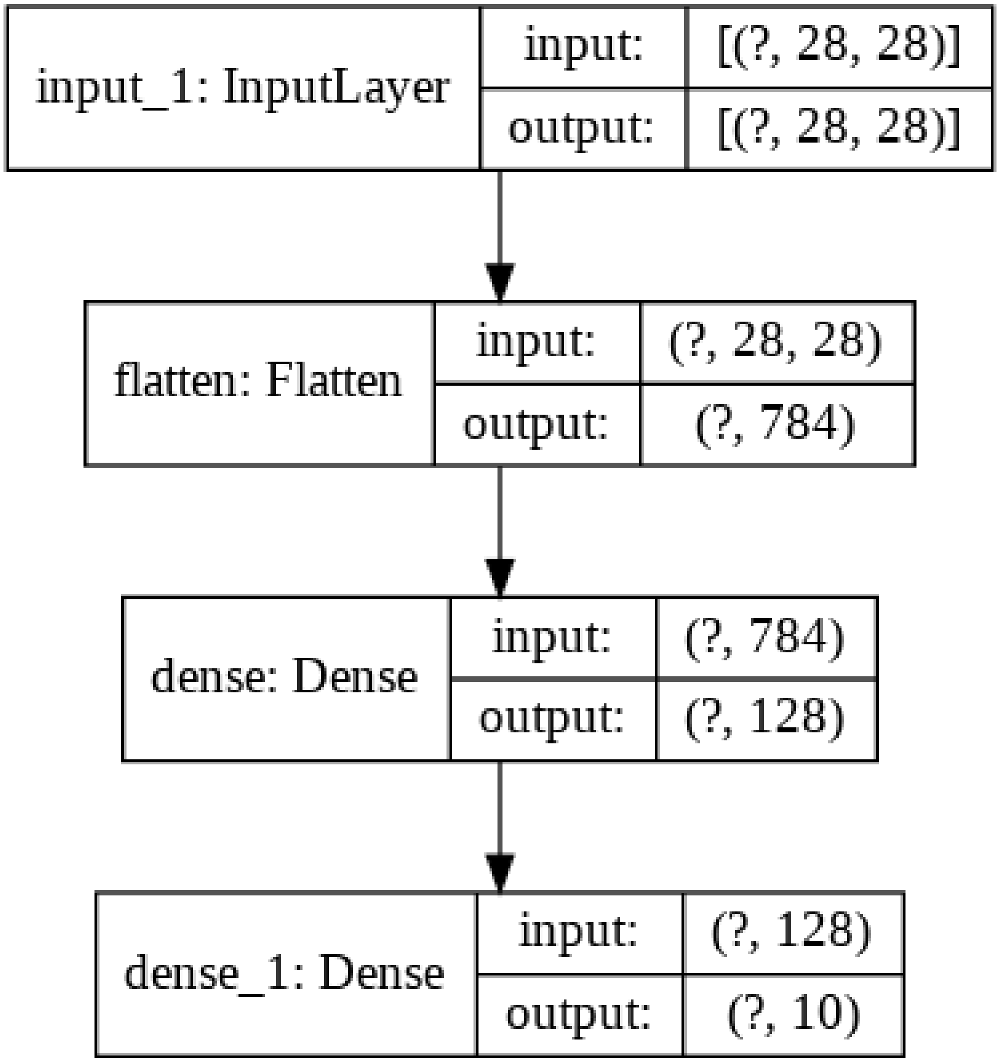
You can choose how to build your model below. Just uncomment which function you'd like to use. You'll notice that the plot will look the same.

[4]:

```
model = build_model_with_functional()
#model = build_model_with_sequential()

# Plot model graph
plot_model(model, show_shapes=True, show_layer_names=True, to_file='model.png')
```

[4]:



1.5 Training the model

Regardless if you built it with the Sequential or Functional API, you'll follow the same steps when training and evaluating your model.

[5]:
```
# prepare fashion mnist dataset
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = mnist.
 ↪load_data()
training_images = training_images / 255.0
test_images = test_images / 255.0

# configure, train, and evaluate the model
```

```
model.compile(optimizer=tf.optimizers.Adam(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(training_images, training_labels, epochs=5)
model.evaluate(test_images, test_labels)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
32768/29515 [=================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 4s 75us/sample - loss: 0.5006 -
accuracy: 0.8231
Epoch 2/5
60000/60000 [==============================] - 4s 70us/sample - loss: 0.3746 -
accuracy: 0.8637
Epoch 3/5
60000/60000 [==============================] - 4s 70us/sample - loss: 0.3347 -
accuracy: 0.8781
Epoch 4/5
60000/60000 [==============================] - 4s 70us/sample - loss: 0.3101 -
accuracy: 0.8862
Epoch 5/5
60000/60000 [==============================] - 4s 70us/sample - loss: 0.2922 -
accuracy: 0.8916
10000/10000 [==============================] - 0s 27us/sample - loss: 0.3510 -
accuracy: 0.8712
```

```
[5]: [0.3510322283267975, 0.8712]
```