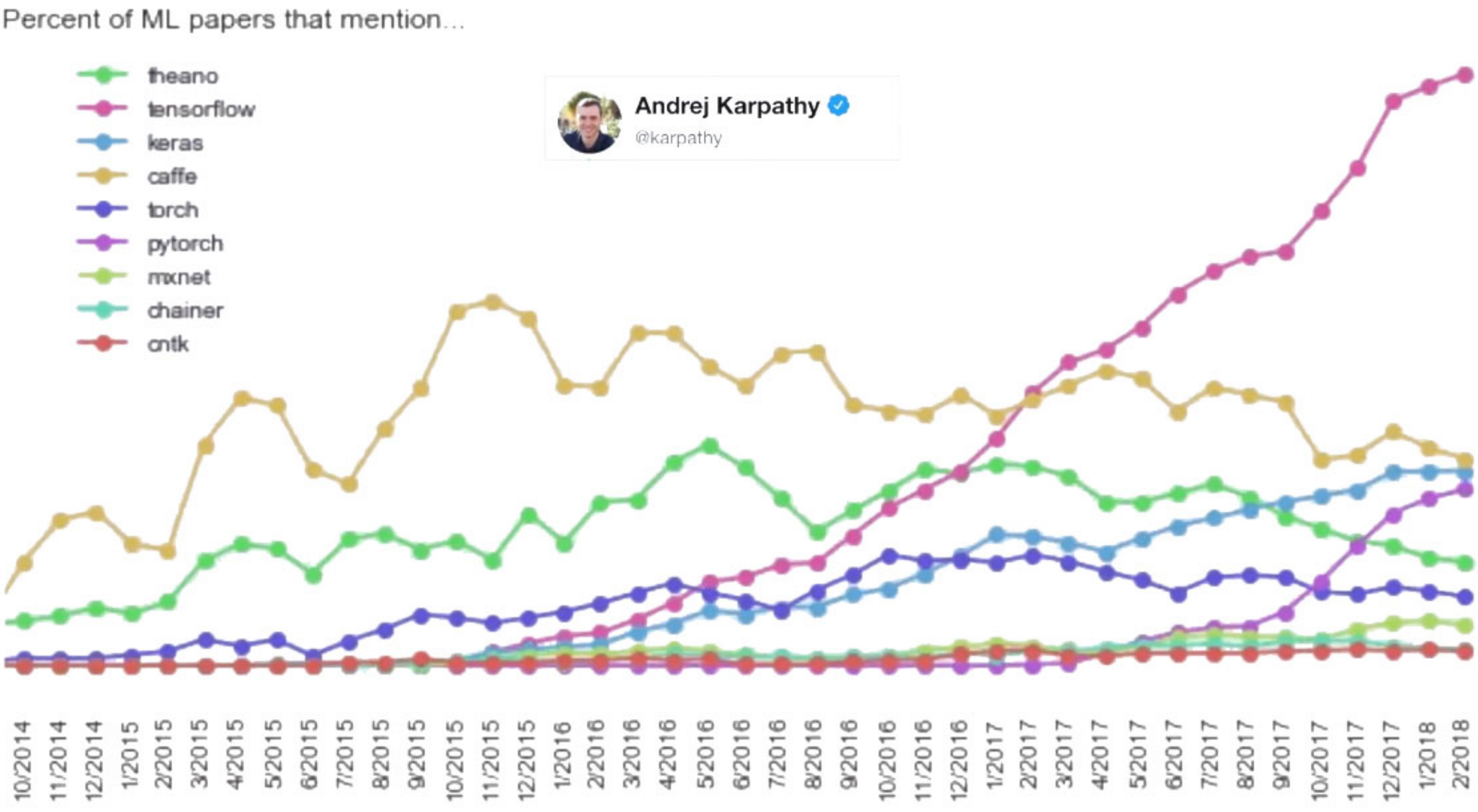
# Overview

TensorFlow began its life in 2011 as DisBelief, an internal, closed source project at Google. DisBelief was a machine learning system that employed deep learning neural networks. This system morphed into TensorFlow, which was released to the developer community under an Apache 2.0 open source license, on November 9, 2015. Version 1.0.0 made its appearance on February 11, 2017. There have been a number of point releases since then that have incorporated a wealth of new features.

TensorFlow quickly rose to become the most widely used machine learning library today. And not without reason.



Google released the TensorFlow 2.0 alpha version in March 2019. The official 2.0 release is expected in Q2 2019. TensorFlow 2.0 promises simplicity

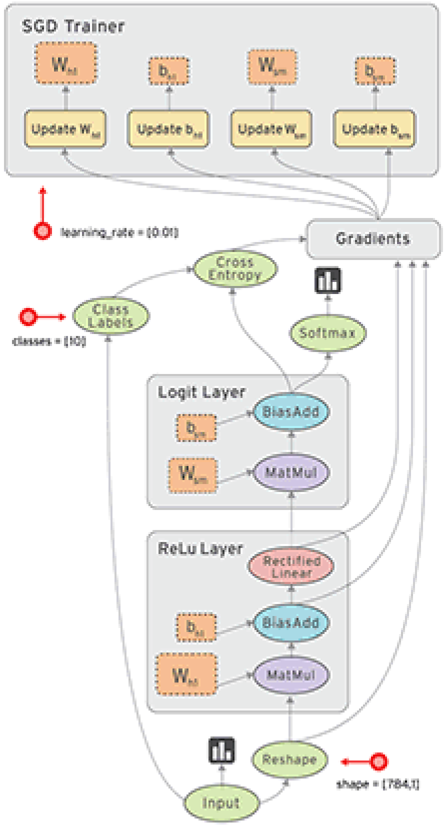
# TensorFlow Ecosystem

- Language Support
- Computation Graph
- Gradients
- CUDA
- Multi-GPU Support
- Eager Execution (> 1.5, > 1.7)
- Debugger
- TensorBoard
- TPU Support
- TensorRT
- `tf.contrib`
- TensorFlow Hub

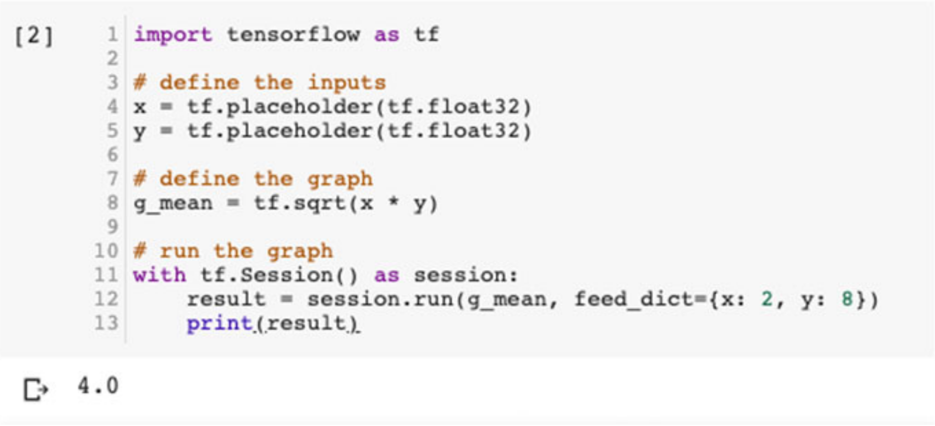
# TensorFlow 1.x [`tf.contrib`]

```
__pycache__/             deprecated/              kafka/                   nn/                      signal/
all_reduce/              distribute/              keras/                   opt/                     slim/
autograph/               distributions/           kernel_methods/          optimizer_v2/            solvers/
batching/                eager/                   kinesis/                 periodic_resample/       sparsemax/
bayesflow/               estimator/               labeled_tensor/          predictor/               specs/
bigtable/                factorization/           layers/                  proto/                   staging/
boosted_trees/           feature_column/          learn/                   quantization/            stat_summarizer/
checkpoint/              ffmpeg/                  legacy_seq2seq/          quantize/                stateless/
cloud/                   framework/               libsvm/                  rate/                    summary/
cluster_resolver/        fused_conv/              linear_optimizer/        receptive_field/         tensor_forest/
cmake/                   gan/                     lite/                    recurrent/               tensorboard/
coder/                   graph_editor/            lookup/                  reduce_slice_ops/        tensorrt/
compiler/                grid_rnn/                losses/                  remote_fused_graph/      testing/
constrained_optimization/ hadoop/                 memory_stats/            resampler/               text/
copy_graph/              hooks/                   meta_graph_transform/    rnn/                     tfprof/
crf/                     ignite/                  metrics/                 rpc/                     timeseries/
cudnn_rnn/               image/                   mixed_precision/         saved_model/             tpu/
data/                    input_pipeline/          model_pruning/           seq2seq/                 training/
decision_trees/          integrate/               nearest_neighbor/        session_bundle/          util/
```

# TensorFlow 1.x [Computation Graph]



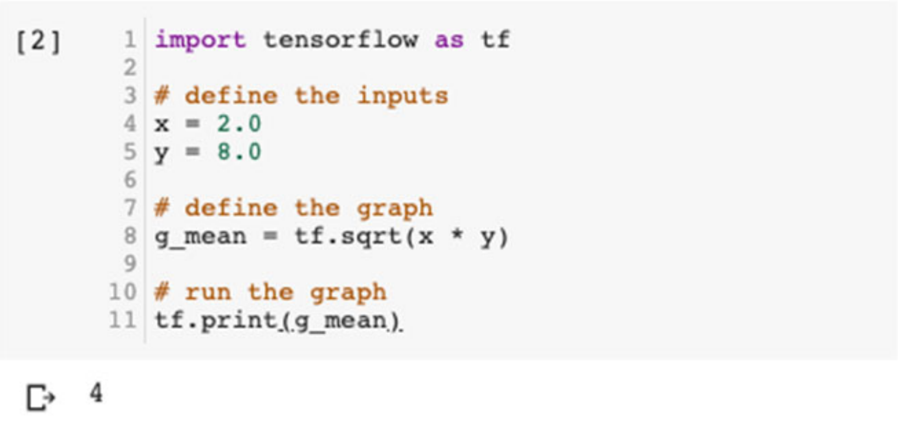
# TensorFlow 1.x Vs. 2.0 [Eager Mode]



[2]

```
import tensorflow as tf

# define the inputs
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# define the graph
g_mean = tf.sqrt(x * y)

# run the graph
with tf.Session() as session:
    result = session.run(g_mean, feed_dict={x: 2, y: 8})
    print(result)
```

```
4.0
```

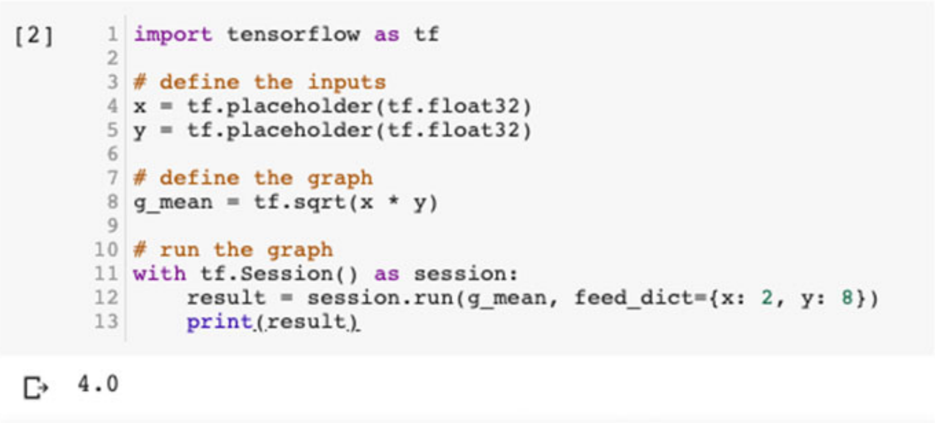


[2]

```
import tensorflow as tf

# define the inputs
x = 2.0
y = 8.0

# define the graph
g_mean = tf.sqrt(x * y)

# run the graph
tf.print(g_mean)
```

```
4
```

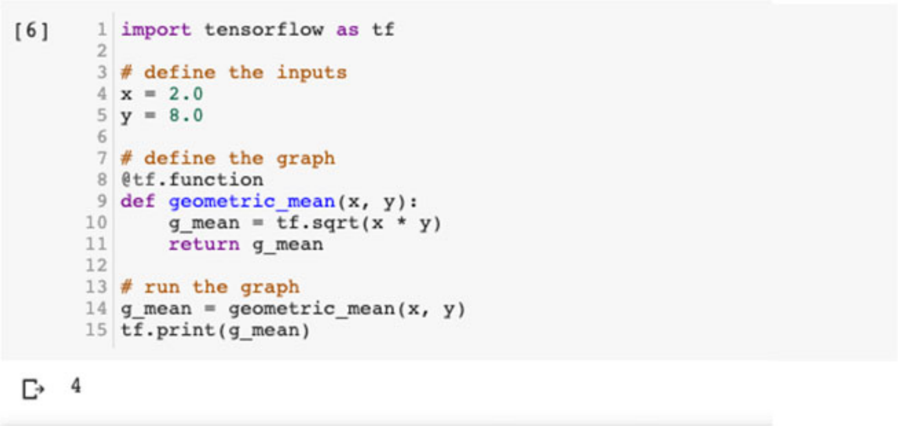
# TensorFlow 1.x Vs. 2.0 [Graph]



[2]

```
import tensorflow as tf

# define the inputs
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# define the graph
g_mean = tf.sqrt(x * y)

# run the graph
with tf.Session() as session:
    result = session.run(g_mean, feed_dict={x: 2, y: 8})
    print(result)
```

```
4.0
```

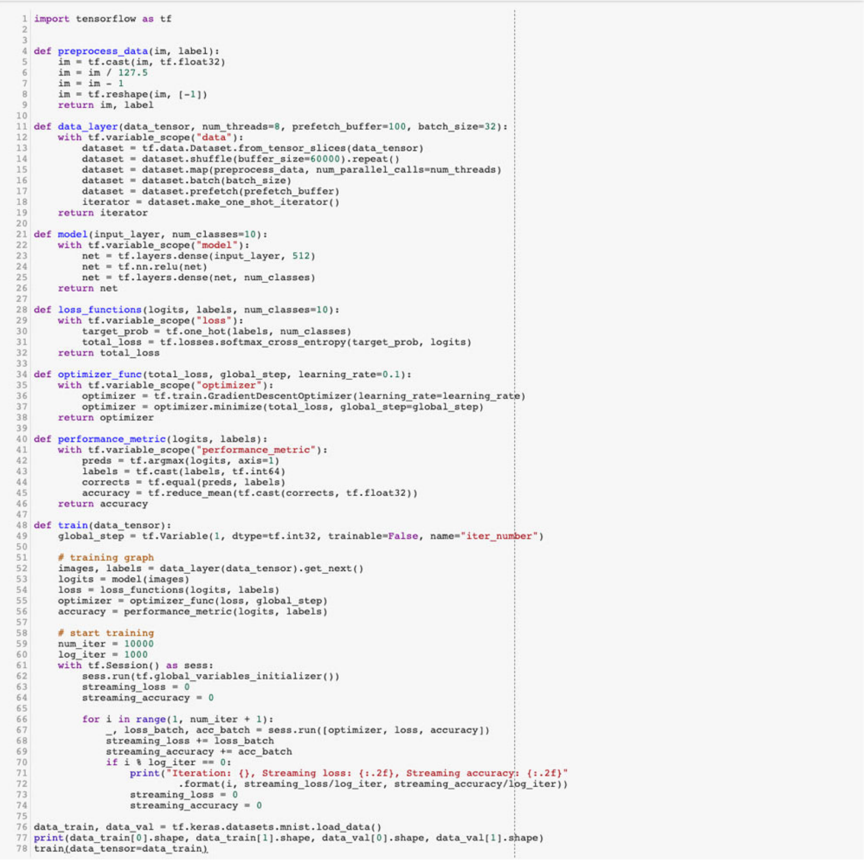


[6]

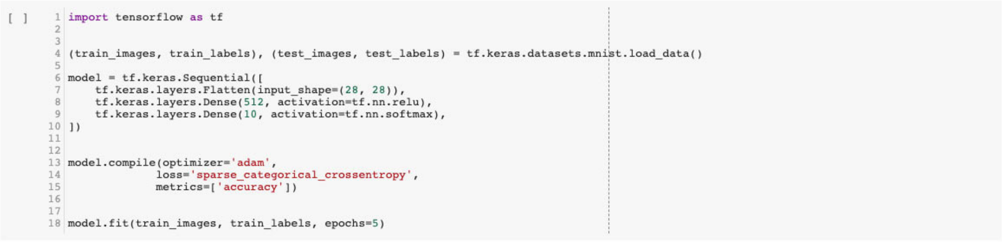
```
import tensorflow as tf

# define the inputs
x = 2.0
y = 8.0

# define the graph
@tf.function
def geometric_mean(x, y):
    g_mean = tf.sqrt(x * y)
    return g_mean

# run the graph
g_mean = geometric_mean(x, y)
tf.print(g_mean)
```

```
4
```

# TensorFlow 1.x Vs. 2.0 [Architecture Layer]

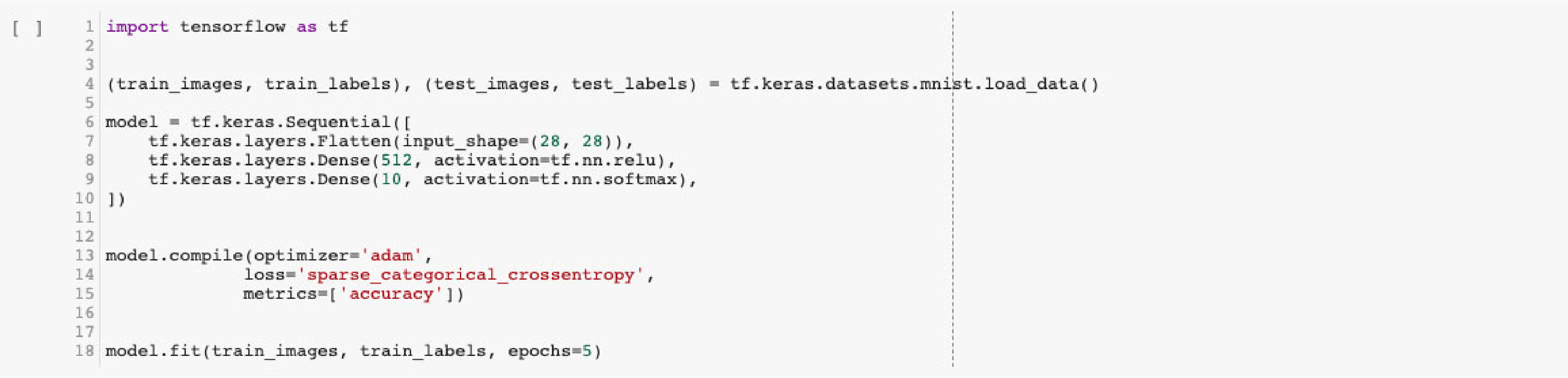


```python
import tensorflow as tf


def preprocess_data(im, label):
    im = tf.cast(im, tf.float32)
    im = im / 127.5
    im = im - 1
    im = tf.reshape(im, [-1])
    return im, label

def data_layer(data_tensor, num_threads=8, prefetch_buffer=100, batch_size=32):
    with tf.variable_scope("data"):
        dataset = tf.data.Dataset.from_tensor_slices(data_tensor)
        dataset = dataset.shuffle(buffer_size=60000).repeat()
        dataset = dataset.map(preprocess_data, num_parallel_calls=num_threads)
        dataset = dataset.batch(batch_size)
        dataset = dataset.prefetch(prefetch_buffer)
        iterator = dataset.make_one_shot_iterator()
    return iterator

def model(input_layer, num_classes=10):
    with tf.variable_scope("model"):
        net = tf.layers.dense(input_layer, 512)
        net = tf.nn.relu(net)
        net = tf.layers.dense(net, num_classes)
    return net

def loss_functions(logits, labels, num_classes=10):
    with tf.variable_scope("loss"):
        target_prob = tf.one_hot(labels, num_classes)
        total_loss = tf.losses.softmax_cross_entropy(target_prob, logits)
    return total_loss

def optimizer_func(total_loss, global_step, learning_rate=0.1):
    with tf.variable_scope("optimizer"):
        optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
        optimizer = optimizer.minimize(total_loss, global_step=global_step)
    return optimizer

def performance_metric(logits, labels):
    with tf.variable_scope("performance_metric"):
        preds = tf.argmax(logits, axis=1)
        labels = tf.cast(labels, tf.int64)
        corrects = tf.equal(preds, labels)
        accuracy = tf.reduce_mean(tf.cast(corrects, tf.float32))
    return accuracy

def train(data_tensor):
    global_step = tf.Variable(1, dtype=tf.int32, trainable=False, name="iter_number")

    # training graph
    images, labels = data_layer(data_tensor).get_next()
    logits = model(images)
    loss = loss_functions(logits, labels)
    optimizer = optimizer_func(loss, global_step)
    accuracy = performance_metric(logits, labels)

    # start training
    num_iter = 10000
    log_iter = 1000
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())
        streaming_loss = 0
        streaming_accuracy = 0

        for i in range(1, num_iter + 1):
            _, loss_batch, acc_batch = sess.run([optimizer, loss, accuracy])
            streaming_loss += loss_batch
            streaming_accuracy += acc_batch
            if i % log_iter == 0:
                print("Iteration: {}, Streaming loss: {:.2f}, Streaming accuracy: {:.2f}"
                        .format(i, streaming_loss/log_iter, streaming_accuracy/log_iter))
                streaming_loss = 0
                streaming_accuracy = 0

data_train, data_val = tf.keras.datasets.mnist.load_data()
print(data_train[0].shape, data_train[1].shape, data_val[0].shape, data_val[1].shape)
train(data_tensor=data_train)
```



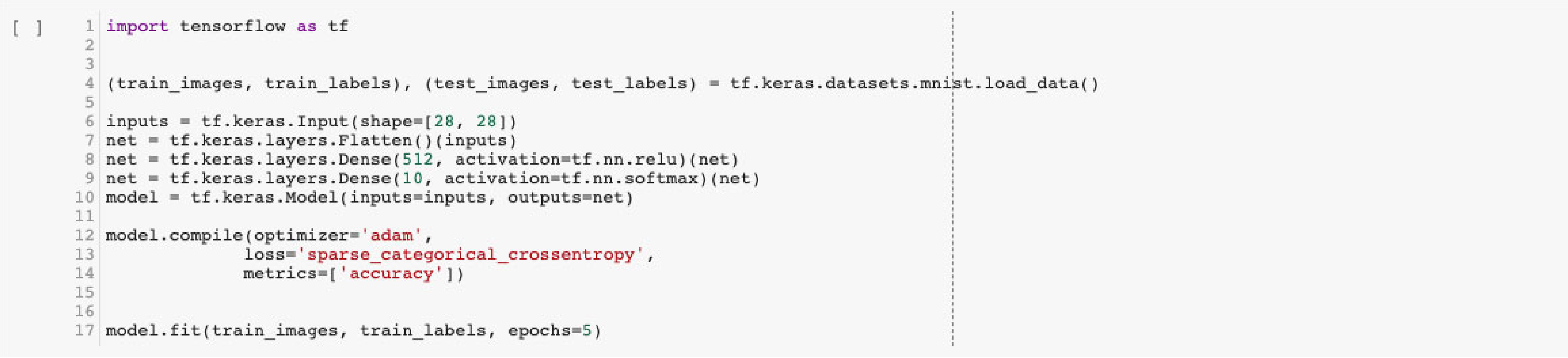
```python
import tensorflow as tf


(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax),
])


model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


model.fit(train_images, train_labels, epochs=5)
```

# TensorFlow 2.0 [Defining Model v1]



```
import tensorflow as tf


(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax),
])


model.compile(optimizer='adam',
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])


model.fit(train_images, train_labels, epochs=5)
```

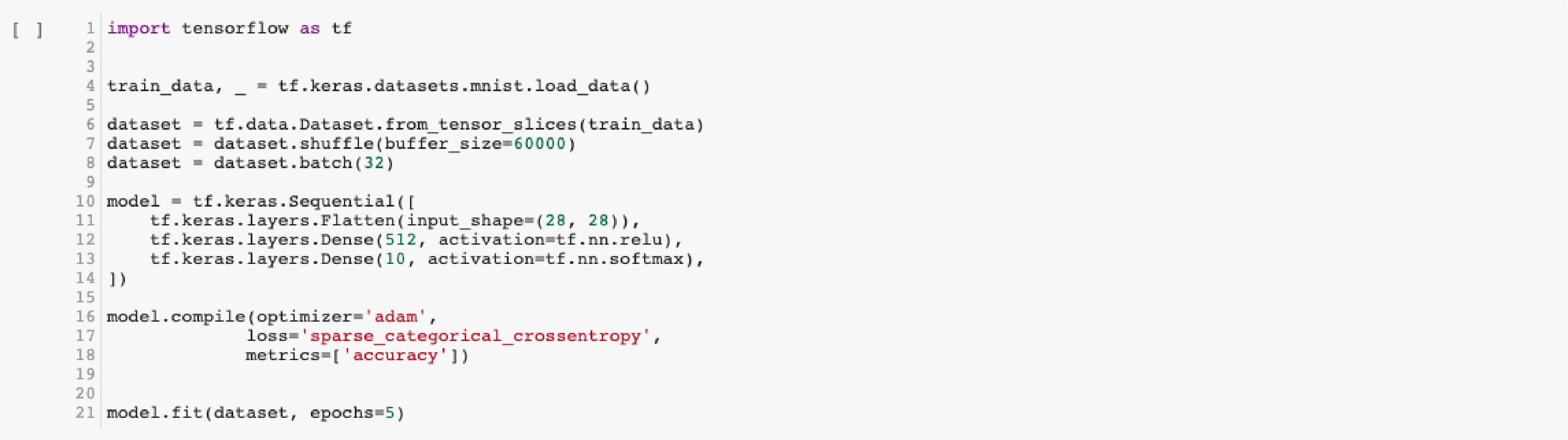
# TensorFlow 2.0 [Defining Model v2]



[ ]

```
import tensorflow as tf


(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()

inputs = tf.keras.Input(shape=[28, 28])
net = tf.keras.layers.Flatten()(inputs)
net = tf.keras.layers.Dense(512, activation=tf.nn.relu)(net)
net = tf.keras.layers.Dense(10, activation=tf.nn.softmax)(net)
model = tf.keras.Model(inputs=inputs, outputs=net)

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


model.fit(train_images, train_labels, epochs=5)
```

# TensorFlow 2.0 [Defining Model v3]



```python
import tensorflow as tf


train_data, _ = tf.keras.datasets.mnist.load_data()

dataset = tf.data.Dataset.from_tensor_slices(train_data)
dataset = dataset.shuffle(buffer_size=60000)
dataset = dataset.batch(32)

model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax),
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])


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