TensorFlow
2.0

Getting started with TensorFlow 2.0

By
Dr Amita Kapoor
and
Ajit Jaokar

Published by Data Science Central

https://www.datasciencecentral.com/

October 2019

Contents

Introduction

In this book, we introduce coding with TensorFlow 2.0. The book is based on three notebooks listed below. We show how to develop with TensorFlow 1.0 and contrast how the same code can be developed in TensorFlow 2.0. The book emphasises the unique features of TensorFlow 2.0.

Notebooks

1. TensorFlow 1.0
2. TensorFlow 2.0
3. TensorFlow datasets

TensorFlow 1.x vs 2.x.ipynb

Overview of changes TensorFlow 1.0 vs TensorFlow 2.0

Earlier this year, Google announced TensorFlow 2.0, it is a major leap from the existing TensorFlow 1.0. The key differences are as follows:

Ease of use: Many old libraries (example tf.contrib) were removed, and some consolidated. For example, in TensorFlow1.x the model could be made using Contrib, layers, Keras or estimators, so many options for the same task confused many new users. TensorFlow 2.0 promotes TensorFlow Keras for model experimentation and Estimators for scaled serving, and the two APIs are very convenient to use.

Eager Execution: In TensorFlow 1.x. The writing of code was divided into two parts: building the computational graph and later creating a session to execute it. this was quite cumbersome, especially if in the big model that you have designed, a small error existed somewhere in the beginning. TensorFlow2.0 Eager Execution is implemented by default, i.e. you no longer need to create a session to run the computational graph, you can see the result of your code directly without the need of creating Session.

Model Building and deploying made easy: With TensorFlow2.0 providing high level TensorFlow Keras API, the user has a greater flexibility in creating the model. One can define model using Keras functional or sequential API. The TensorFlow Estimator API allows one to run model on a local host or on a distributed multi-server environment without changing your model. Computational graphs are powerful in terms of performance, in TensorFlow 2.0 you can use the decorator **tf.function** so that the following function block is run as a single graph. This is done via the powerful Autograph feature of TensorFlow 2.0. This allows users to optimize the function and increase portability. And the best part you can write the function using natural Python syntax. Effectively, you can use the decorator tf.function to turn plain Python code into graph. While the decorator @tf.function applies to the function block immediately following it, any functions ***called*** by it will be executed in graph mode as well. Thus, in TensorFlow 2.0, users should refactor their code into smaller functions which are called as needed. In general, it's not necessary to decorate each of these smaller functions with tf.function; only use tf.function to decorate high-level computations - for example, one step of training, or the forward pass of your model. (source stack overflow and TF2 documentation)

To expand this idea, In TensorFlow 1.x we needed to build the computational graph. TensorFlow 2.0 does not build graph by default. However, as every Machine Learning engineer knows, graphs are good for speed. TensorFlow 2.0 provides the user to create a callable graph using a python function @tf.function. The tf.function() will create a separate graph for every unique set of input shapes and datatypes. In the example below we will have three separate graphs created, one for each input datatype.

```
@tf.function
def f(x): return tf.add(x, 1.)
scalar = tf.constant(1.0)
vector = tf.constant([1.0, 1.0])
matrix = tf.constant([[3.0]])
print(f(scalar))
print(f(vector))
print(f(matrix))
```

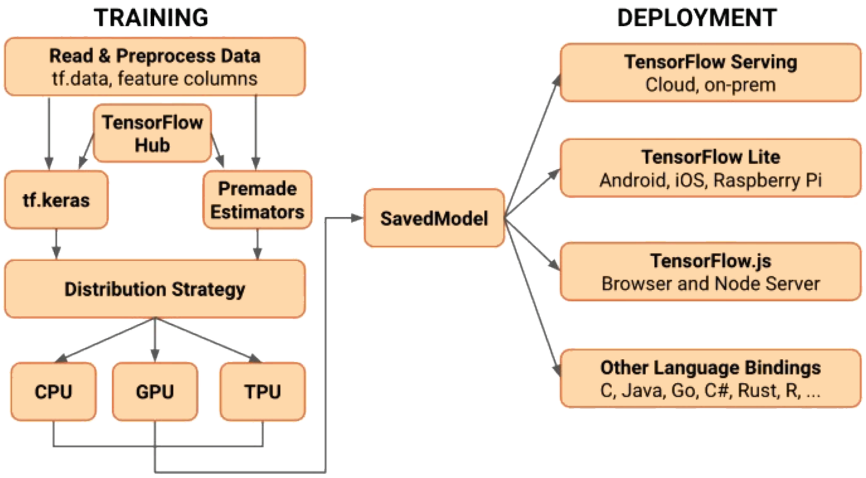
The Data pipeline simplified: TensorFlow2.0 has a separate module TensorFlow DataSets that can be used to operate with the model in more elegant way. Not only it has a large range of existing datasets, making your job of experimenting with a new architecture easier – it also has well defined way to add your data to it.

In TensorFlow 1.x for building a model we would first need to declare placeholders. These were the dummy variables which will later (in the session) used to feed data to the model. There were many built-in APIs for building the layers like tf.contrib, tf.layers and tf.keras, one could also build layers by defining the actual mathematical operations.

TensorFlow 2.0 you can build your model defining your own mathematical operations, as before you can use math module (tf.math) and linear algebra (tf.linalg) module. However, you can take advantage of the high level Keras API and tf.layers module. ***The important part is we do not need to define placeholders any more.***

Simplified conceptual diagram for TensorFlow 2.0

A simplified, conceptual diagram as shown below for TensorFlow 2.0



Source: https://medium.com/tensorflow/whats-coming-in-tensorflow-2-0-d3663832e9b8

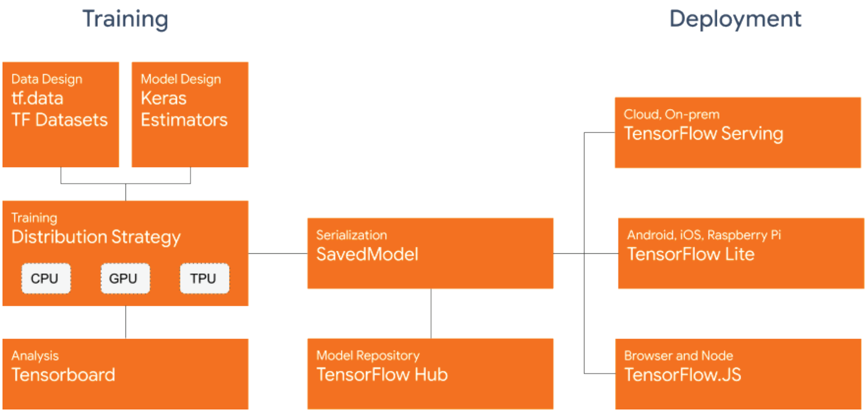
Some important points are:

- Load your data using data. Training data is read using input pipelines which are created using tf.data.
- Use TensorFlow Dataset to get a large variety of datasets to train your model.
- Build, train and validate your model with keras, or use Estimators API.
- TensorFlow Hub in the TensorFlow ecosystem contains a large number of pretrained models, using the standard interface, you can import any of the models from TensorFlow

Hub and either train it from scratch or fine tune it for your data using transfer learning technique.

- Run and debug with eager execution, then use function for the benefits of graphs.
- Use Distribution Strategies for distributed training. For large ML training tasks, the Distribution Strategy API makes it easy to distribute and train models on different hardware configurations without changing the model definition. You can distribute your training load to a range of hardware accelerators like CPUs, GPUs, and TPUs
- Although this API supports a variety of cluster configurations, templates to deploy training on Kubernetes clusters in on-prem or cloud environments are provided.
- Export to SavedModel. TensorFlow will standardize on SavedModel as an interchange format for TensorFlow Serving, TensorFlow Lite, TensorFlow.js, TensorFlow Hub, and more.
- Once you've trained and saved your model, you can execute it directly in your application or serve it using one of the deployment libraries: TensorFlow Serving: A TensorFlow library allowing models to be served over HTTP/REST or gRPC/Protocol Buffers. TensorFlow Lite: TensorFlow's lightweight solution for mobile and embedded devices provides the capability to deploy models on Android, iOS and embedded systems like a Raspberry Pi and Edge TPUs. js: Enables deploying models in JavaScript environments, such as in a web browser or server side through Node.js. TensorFlow.js also supports defining models in JavaScript and training directly in the web browser using a Keras-like API.

Coding with TensorFlow 2.0



TensorFlow 2.0 offers many performance improvements on GPUs. TensorFlow 2.0 delivers up to 3x faster training performance using mixed precision on Volta and Turing GPUs with a few lines of code, used for example in ResNet-50 and BERT. TensorFlow 2.0 is tightly integrated with TensorRT and uses an improved API to deliver better usability and high performance during inference on NVIDIA T4 Cloud GPUs on Google Cloud.

Above section adapted from
https://medium.com/tensorflow/whats-coming-in-tensorflow-2-0-d3663832e9b8
and https://medium.com/tensorflow/tensorflow-2-0-is-now-available-57d706c2a9ab

Observations

On one hand, Tensorflow 2.0 does not feel new. Probably because even in the age of Tensorflow 1.0, almost everyone was using keras! Keras is now central to Tensorflow 2.0 but Tensorflow 2.0 has much more features as we see above.

TensorFlow1.0

Creating computational graph and running a session.

```
[ ]
!pip install tensorflow-gpu==1.14

[ ]
# Import Tensorflow and check the version
import tensorflow as tf
print(tf.__version__)

1.14.0

[ ]
# define the computational graph
a = tf.constant("Hello world!")
[ ]
# run the computational graph in session
with tf.Session() as sess:
  result = sess.run(a)
  print(result)

'Hello world!'

Building a model
[ ]
# Getting data
from sklearn.datasets import load_iris
from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import OneHotEncoder

iris_data = load_iris() # load the iris dataset

x = iris_data.data
```

```
y_ = iris_data.target.reshape(-1, 1) # Convert data
to a single column

# One Hot encode the class labels
encoder = OneHotEncoder(sparse=False)
y = encoder.fit_transform(y_)

train_x, test_x, train_y, test_y = train_test_split(x,
y, test_size=0.20)

/usr/local/lib/python3.6/dist-
packages/sklearn/preprocessing/_encoders.py:415:
```

In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, then you can now use the OneHotEncoder directly.

```
warnings.warn(msg, FutureWarning)


[ ]
n_input = 4
n_output = 3
n_hidden = 10

#hyperparameter
learning_rate = 0.01
training_epochs = 2000
display_steps = 200
[ ]
# Create placeholder and Variables for the input
and weights

#Graph Nodes
X = tf.placeholder("float", [None, n_input])
Y = tf.placeholder("float", [None, n_output])

#Weights and Biases
weights = {
  "hidden" : tf.Variable(tf.random_normal([n_input,
n_hidden]), name="weight_hidden"),
  "output" :
tf.Variable(tf.random_normal([n_hidden, n_output]),
name="weight_output")
}

bias = {
  "hidden" : tf.Variable(tf.random_normal([n_hidden]),
name="bias_hidden"),
  "output" : tf.Variable(tf.random_normal([n_output]),
name="bias_output")
}

[ ]
def model(x, weights, bias):
  layer_1 = tf.add(tf.matmul(x, weights["hidden"]),
bias["hidden"])
  layer_1 = tf.nn.relu(layer_1)

  output_layer = tf.matmul(layer_1, weights["output"])
+ bias["output"]
  return output_layer
[ ]
perceptron = model(X, weights, bias)
[ ]
#Define loss and optimizer
cost =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_log
its(logits=perceptron, labels=Y))
optimizer =
tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.

```
[ ]
#Initializing global variables
init = tf.global_variables_initializer()
```

```
with tf.Session() as sess:
  sess.run(init)

  for epoch in range(training_epochs):
    _, c = sess.run([optimizer, cost], feed_dict={X:
train_x, Y: train_y})
    if(epoch + 1) % display_steps == 0:
      print ("Epoch: ", (epoch+1), "Cost: ", c)

  print("Optimization Finished!")

  test_result = sess.run(perceptron, feed_dict={X:
test_x})
  correct_pred = tf.equal(tf.argmax(test_result,
1), tf.argmax(test_y, 1))

  accuracy = tf.reduce_mean(tf.cast(correct_pred,
"float"))
  print ("Accuracy:", accuracy.eval({X: test_x, Y:
test_y}))
```

```
Epoch:  200 Cost:  0.26588085
Epoch:  400 Cost:  0.14439923
Epoch:  600 Cost:  0.10045961
Epoch:  800 Cost:  0.08092845
Epoch:  1000 Cost:  0.070696324
Epoch:  1200 Cost:  0.06470656
Epoch:  1400 Cost:  0.060857367
Epoch:  1600 Cost:  0.05816547
Epoch:  1800 Cost:  0.056141052
Epoch:  2000 Cost:  0.054527108
Optimization Finished!
Accuracy: 1.0
```

TensorFlow2.0

```
[ ] !pip install tensorflow-gpu==2.0.0-rc1

[ ]
# Import Tensorflow and check the version
import tensorflow as tf
print(tf.__version__)


[ ]
a = tf.constant("Hello world!")
print(a)


Autograph and tf.function()
[ ]
@tf.function
def f(x): return tf.add(x, 1.)

scalar = tf.constant(1.0)
vector = tf.constant([1.0, 1.0])
matrix = tf.constant([[3.0]])

print(f(scalar))
print(f(vector))
print(f(matrix))


Building a model
[ ]
n_input = 4
n_output = 3
n_hidden = 10

#hyperparameter
learning_rate = 0.01
training_epochs = 2000
```

```
display_steps = 200
[ ]
# Getting data
from sklearn.datasets import load_iris
from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import OneHotEncoder

iris_data = load_iris() # load the iris dataset

x = iris_data.data
y_ = iris_data.target.reshape(-1, 1) # Convert data
to a single column

# One Hot encode the class labels
encoder = OneHotEncoder(sparse=False)
y = encoder.fit_transform(y_)

train_x, test_x, train_y, test_y = train_test_split(x,
y, test_size=0.20)
```

```
 /usr/local/lib/python3.6/dist-packages/sklearn/
preprocessing/_encoders.py:415: FutureWarning:
```

The handling of integer data will change in version 0.22. Currently, the categories are determined based on the range [0, max(values)], while in the future they will be determined based on the unique values.

If you want the future behaviour and silence this warning, you can specify "categories='auto'".

In case you used a LabelEncoder before this OneHotEncoder to convert the categories to integers, then you can now use the OneHotEncoder directly.

```
warnings.warn(msg, FutureWarning)
```

```
[ ]
# Build the model

model = tf.keras.Sequential()

model.add(tf.keras.layers.Dense(n_hidden,
input_shape=(n_input,), activation='relu',
name='fc1'))
model.add(tf.keras.layers.Dense(n_output,
activation='softmax', name='output'))

# Adam optimizer with learning rate of 0.001
optimizer = tf.keras.optimizers.Adam(lr=0.001)
model.compile(optimizer,
loss='categorical_crossentropy', metrics=['accuracy'])

print('Neural Network Model Summary: ')
print(model.summary())
 Neural Network Model Summary:
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
fc1 (Dense)                  (None, 10)                50
_________________________________________________________________
output (Dense)               (None, 3)                 33
=================================================================
Total params: 83
Trainable params: 83
Non-trainable params: 0
_________________________________________________________________
None
```

```
[ ]
# Train the model
model.fit(train_x, train_y, verbose=2, batch_size=5,
epochs=200)

# Test on unseen data

results = model.evaluate(test_x, test_y)
```

```
Train on 120 samples
Epoch 1/200
- 0s 7ms/sample - loss: 0.0340 - accuracy: 1.0000
[ ]
print('Final test set loss: {:4f}'.format(results[0]))
print('Final test set accuracy: {:4f}'.format(results[1]))


[ ]
!pip install tensorflow-datasets
Collecting tensorflow-datasets
  Downloading
Installing collected packages: tensorflow-datasets
Successfully installed tensorflow-datasets-1.2.0

[ ]
import tensorflow_datasets as tfds
[ ]
tfds.list_builders()

['abstract_reasoning',
 'aflw2k3d',
 'amazon_us_reviews',
 'bair_robot_pushing_small',
 'bigearthnet',
 'binarized_mnist',
 'binary_alpha_digits',
 'caltech101',
 'caltech_birds2010',
 'caltech_birds2011',
 'cats_vs_dogs',
 'celeb_a',
 'celeb_a_hq',
 'chexpert',
 'cifar10',
 'cifar100',
 'cifar10_corrupted',
 'clevr',
 'cnn_dailymail',
 'coco',
 'coco2014',
 'coil100',
 'colorectal_histology',
 'colorectal_histology_large',
 'curated_breast_imaging_ddsm',
 'cycle_gan',
 'deep_weeds',
 'definite_pronoun_resolution',
 'diabetic_retinopathy_detection',
 'downsampled_imagenet',
 'dsprites',
 'dtd',
 'dummy_dataset_shared_generator',
 'dummy_mnist',
 'emnist',
 'eurosat',
 'fashion_mnist',
 'flores',
 'food101',
 'gap',
 'glue',
 'groove',
 'higgs',
 'horses_or_humans',
 'image_label_folder',
 'imagenet2012',
 'imagenet2012_corrupted',
 'imdb_reviews',
 'iris',
 'kitti',
 'kmnist',
 'lfw',
 'lm1b',
 'lsun',
 'mnist',
 'mnist_corrupted',
 'moving_mnist',
 'multi_nli',
 'nsynth',
 'omniglot',
 'open_images_v4',
 'oxford_flowers102',
 'oxford_iiit_pet',
 'para_crawl',
 'patch_camelyon',
 'pet_finder',
 'quickdraw_bitmap',
 'resisc45',
 'rock_paper_scissors',
 'rock_you',
 'scene_parse150',
 'shapes3d',
 'smallnorb',
 'snli',
 'so2sat',
 'squad',
 'stanford_dogs',
 'stanford_online_products',
 'starcraft_video',
 'sun397',
 'super_glue',
 'svhn_cropped',
 'ted_hrlr_translate',
 'ted_multi_translate',
 'tf_flowers',
 'titanic',
 'trivia_qa',
 'uc_merced',
 'ucf101',
 'visual_domain_decathlon',
 'voc2007',
 'wikipedia',
 'wmt14_translate',
 'wmt15_translate',
 'wmt16_translate',
 'wmt17_translate',
 'wmt18_translate',
 'wmt19_translate',
 'wmt_t2t_translate',
 'wmt_translate',
 'xnli']
```

```
[ ]
iris = tfds.load(name="iris", split=None)
 Downloading and preparing dataset iris (4.44 KiB)
to /root/tensorflow_datasets/iris/1.0.0...
HBox(children=(IntProgress(value=1, bar_style='info',
description='Dl Completed...', max=1,
style=ProgressStyl...
HBox(children=(IntProgress(value=1,
bar_style='info', description='Dl Size...', max=1,
style=ProgressStyle(des...
/usr/local/lib/python3.6/dist-packages/urllib3/
connectionpool.py:847: InsecureRequestWarning:
Unverified HTTPS request is being made. Adding
certificate verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/advanced-
usage.html#ssl-warnings
  InsecureRequestWarning)
HBox(children=(IntProgress(value=1, bar_style='info',
max=1), HTML(value='')))
HBox(children=(IntProgress(value=0,
description='Shuffling...', max=1,
style=ProgressStyle(description_width='...
WARNING:tensorflow:From
/usr/local/lib/python3.6/dist-packages/
tensorflow_datasets/core/file_format_adapter.py:209
: tf_record_iterator (from
tensorflow.python.lib.io.tf_record) is deprecated
and will be removed in a future version.
Instructions for updating:
Use eager execution and:
`tf.data.TFRecordDataset(path)`
WARNING:tensorflow:From
/usr/local/lib/python3.6/dist-packages/
tensorflow_datasets/core/file_format_adapter.py:209
: tf_record_iterator (from
tensorflow.python.lib.io.tf_record) is deprecated
and will be removed in a future version.
Instructions for updating:
Use eager execution and:
`tf.data.TFRecordDataset(path)`
HBox(children=(IntProgress(value=1,
bar_style='info', description='Reading...', max=1,
style=ProgressStyle(des...
HBox(children=(IntProgress(value=0,
description='Writing...', max=150,
style=ProgressStyle(description_width='...
WARNING:absl:Warning: Setting shuffle_files=True
because split=TRAIN and shuffle_files=None. This
behavior will be deprecated on 2019-08-06, at which
point shuffle_files=False will be the default for
all splits.
```

```
Dataset iris downloaded and prepared to
/root/tensorflow_datasets/iris/1.0.0. Subsequent
calls will reuse this data.

[ ]
```

TensorFlowDataset.ipynb_

```
[ ]
# Install TensorFlow 2.0
!pip install -q tensorflow-gpu==2.0.0-rc
```

TensorFlow Datasets

Data as we all know is the new "gold", no matter what application you have in mind it can be improved upon by sufficient data. Getting data used to be one of the most difficult task in AI/ML, but thanks to Google's TensorFlow Dataset it is not so any more.

In this post you will learn about TensorFlow Dataset and how to include it in your ML pipeline.

Before we start you should ensure that you have the laters version of TensorFlow installed. We will need to install TensorFlow Dataset as well:

```
pip install tensorflow-datasets
```

```
[ ]
!pip install tensorflow-datasets
```

Now that TensorFlow DataSet is installed let us import it and get a list of all the available datasets: we can do this by using list_builders() function. This results in generating a list of string where each string is a dataset. For example, the iris dataset is listed in the list as string 'iris'.

The TensorFlow Dataset works over TensorFlow, so in order to use it you will need to import TensorFlow as well.

```
[ ]
import tensorflow as tf
import tensorflow_datasets as tfds
tfds.list_builders()
['abstract_reasoning',
 'aflw2k3d',
 'amazon_us_reviews',
 'bair_robot_pushing_small',
 'bigearthnet',
 'binarized_mnist',
 'binary_alpha_digits',
 'caltech101',
 'caltech_birds2010',
 'caltech_birds2011',
 'cats_vs_dogs',
 'celeb_a',
 'celeb_a_hq',
 'chexpert',
 'cifar10',
 'cifar100',
 'cifar10_corrupted',
 'clevr',
 'cnn_dailymail',
 'coco',
 'coco2014',
 'coil100',
 'colorectal_histology',
 'colorectal_histology_large',
 'curated_breast_imaging_ddsm',
 'cycle_gan',
 'deep_weeds',
 'definite_pronoun_resolution',
 'diabetic_retinopathy_detection',
 'downsampled_imagenet',
 'dsprites',
 'dtd',
 'dummy_dataset_shared_generator',
 'dummy_mnist',
 'emnist',
 'eurosat',
 'fashion_mnist',
 'flores',
 'food101',
 'gap',
 'glue',
 'groove',
 'higgs',
 'horses_or_humans',
 'image_label_folder',
 'imagenet2012',
 'imagenet2012_corrupted',
 'imdb_reviews',
 'iris',
 'kitti',
 'kmnist',
 'lfw',
 'lm1b',
 'lsun',
 'mnist',
 'mnist_corrupted',
 'moving_mnist',
 'multi_nli',
 'nsynth',
 'omniglot',
 'open_images_v4',
 'oxford_flowers102',
 'oxford_iiit_pet',
 'para_crawl',
 'patch_camelyon',
 'pet_finder',
 'quickdraw_bitmap',
 'resisc45',
 'rock_paper_scissors',
 'rock_you',
 'scene_parse150',
 'shapes3d',
 'smallnorb',
 'snli',
 'so2sat',
 'squad',
 'stanford_dogs',
 'stanford_online_products',
 'starcraft_video',
 'sun397',
 'super_glue',
 'svhn_cropped',
 'ted_hrlr_translate',
 'ted_multi_translate',
 'tf_flowers',
 'titanic',
 'trivia_qa',
 'uc_merced',
 'ucf101',
 'visual_domain_decathlon',
 'voc2007',
 'wikipedia',
 'wmt14_translate',
 'wmt15_translate',
 'wmt16_translate',
 'wmt17_translate',
 'wmt18_translate',
 'wmt19_translate',
 'wmt_t2t_translate',
 'wmt_translate',
 'xnli']
```

Each dataset can be loaded using load function with its name as string, you can specify the complete data, or just specific set. There are various options that you can specify with the load function, for complete list refer here.

Let us try loading one of the data from the list, we chose the one most familiar: MNIST dataset. The load function returns the dataset requested and if with_info parameter is set to True then also information about the dataset.

```
[ ]
data, info = tfds.load(name='fashion_mnist',
as_supervised=True, split=None, with_info=True)
```

If you print the info it give you details about the dataset, the type of dataset, number of training, test samples. Its original source and even how to cite it.

```
[ ]
print(info)
tfds.core.DatasetInfo(
    name='fashion_mnist',
    version=1.0.0,
    description='Fashion-MNIST is a dataset of
Zalando's article images consisting of a training
set of 60,000 examples and a test set of 10,000
examples. Each example is a 28x28 grayscale image,
associated with a label from 10 classes.',
    urls=['https://github.com/zalandoresearch/
fashion-mnist'],
    features=FeaturesDict({
       'image': Image(shape=(28, 28, 1), dtype=tf.uint8),
       'label': ClassLabel(shape=(),
dtype=tf.int64, num_classes=10),
    }),
    total_num_examples=70000,
    splits={
        'test': 10000,
        'train': 60000,
    },
   supervised_keys=('image', 'label'),
   citation="""@article{DBLP:journals/corr/abs-1708-07747,
      author    = {Han Xiao and
                   Kashif Rasul and
                   Roland Vollgraf},
      title     = {Fashion-MNIST: a Novel Image
Dataset for Benchmarking Machine Learning
                   Algorithms},
      journal   = {CoRR},
      volume    = {abs/1708.07747},
      year      = {2017},
      url       =
{http://arxiv.org/abs/1708.07747},
      archivePrefix = {arXiv},
      eprint    = {1708.07747},
      timestamp = {Mon, 13 Aug 2018 16:47:27 +0200},
      biburl    =
{https://dblp.org/rec/bib/journals/corr/abs-1708-07747},
      bibsource = {dblp computer science
bibliography, https://dblp.org}
    }""",
    redistribution_info=,
)
```

Let us now get training and test data set

```
[ ]
train, test = data['train'], data['test']
```

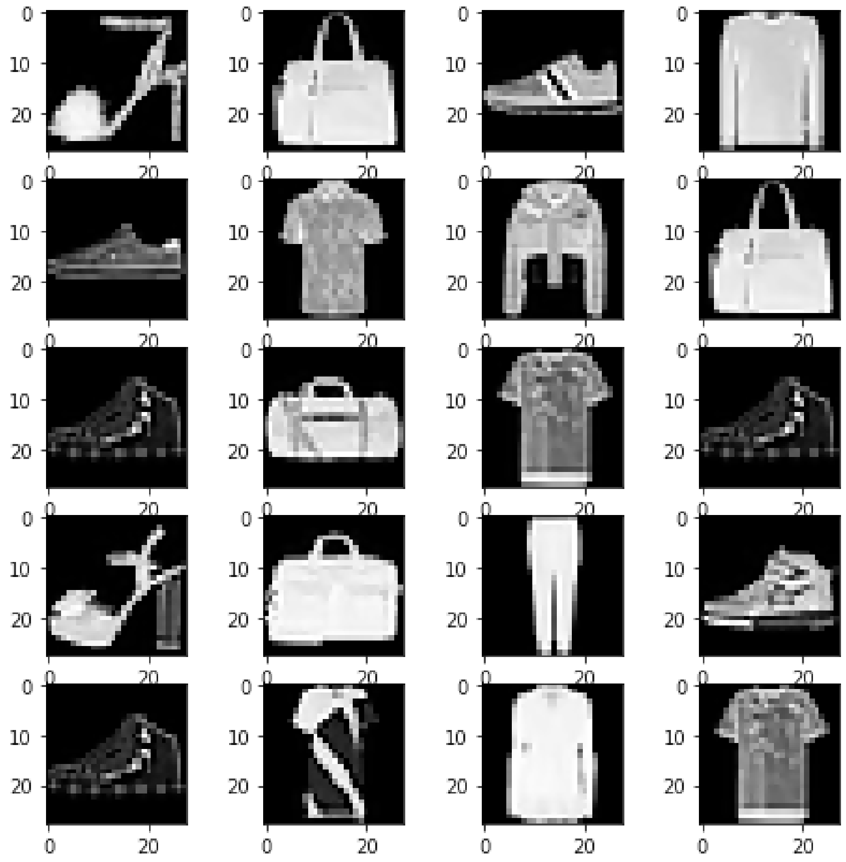
For the ML pipeline we need to get data in batches, preferrable shuffled. To get a stream of data repeatedly we can use functions shuffle() and batch(). Also the images need to be normalized, so we define a function normalize and define our training hyperparameters.

```
[ ]
BUFFER_SIZE = 10 # Use a much larger value for real
code.
BATCH_SIZE = 64
NUM_EPOCHS = 5


def normalize(image, label):
  image = tf.cast(image, tf.float32)
  image /= 255

  return image, label

train_data = train.map(normalize).
shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
test_data = test.map(normalize).batch(BATCH_SIZE)

STEPS_PER_EPOCH = 5

train_data = train_data.take(STEPS_PER_EPOCH)
test_data = test_data.take(STEPS_PER_EPOCH)
```

You can iterate over the entire dataset using next(iter(train_data)). The code will generate data in batches defined by BATCH_SIZE.

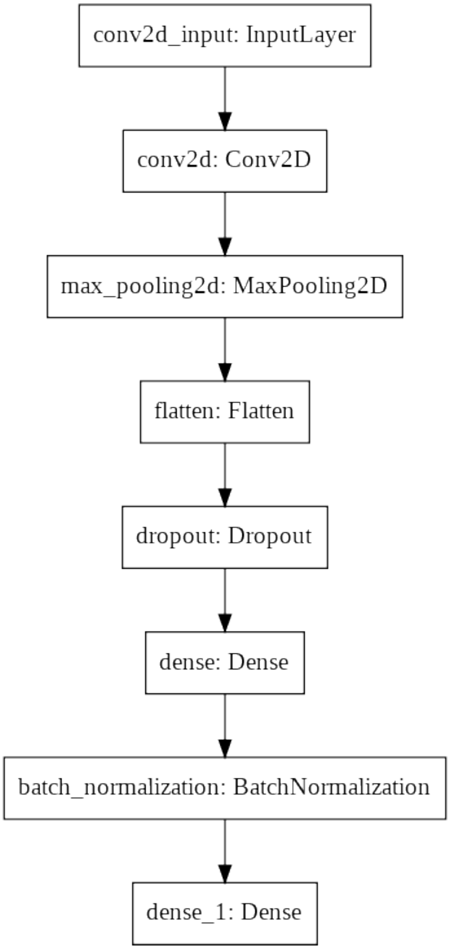
```
[ ]
image_batch, label_batch = next(iter(train_data))

[ ]
import numpy as np
import matplotlib.pyplot as plt
idx = np.random.randint(0, BATCH_SIZE, size =30 )
```

```
fig=plt.figure(figsize=(8, 8))
columns = 4
rows = 5
for i in range(1, columns*rows +1):
    fig.add_subplot(rows, columns, i)
    plt.imshow(image_batch[idx[i],:,:,0], cmap='gray')
plt.show()
```



And now let us build a simple model and train it for the Fashion-MNIST data.

```
[ ]

model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, 3, activation='relu',

kernel_regularizer=tf.keras.regularizers.l2(0.02),
                           input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dense(10, activation='softmax')
])
```



```
# Model is the full model w/o custom layers
model.compile(optimizer='adam',

loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_data, epochs=NUM_EPOCHS)
loss, acc = model.evaluate(test_data)

print("Loss {}, Accuracy {}".format(loss, acc))

Epoch 1/5
WARNING:tensorflow:Entity <function
Function._initialize_uninitialized_variables.<locals
>.initialize_variables at 0x7ff3f28c0400> could not
be transformed and will be executed as-is. Please
report this to the AutoGraph team. When filing the
bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: module 'gast' has no attribute 'Num'
WARNING:tensorflow:Entity <function
Function._initialize_uninitialized_variables.<locals
>.initialize_variables at 0x7ff3f28c0400> could not
be transformed and will be executed as-is. Please
report this to the AutoGraph team. When filing the
bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: module 'gast' has no attribute 'Num'
WARNING: Entity <function
Function._initialize_uninitialized_variables.<locals
>.initialize_variables at 0x7ff3f28c0400> could not
be transformed and will be executed as-is. Please
report this to the AutoGraph team. When filing the
bug, set the verbosity to 10 (on Linux, `export
AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: module 'gast' has no attribute 'Num'
5/5 [==============================] - 2s 417ms/step
- loss: 1.6265 - accuracy: 0.4281
Epoch 2/5
```

```
5/5 [==============================] - 0s 51ms/step
- loss: 0.8640 - accuracy: 0.7469
Epoch 3/5
5/5 [==============================] - 0s 49ms/step
- loss: 0.7518 - accuracy: 0.7937
Epoch 4/5
5/5 [==============================] - 0s 49ms/step
- loss: 0.6016 - accuracy: 0.8500
Epoch 5/5
5/5 [==============================] - 0s 53ms/step
- loss: 0.5105 - accuracy: 0.8781
5/5 [==============================] - 0s 51ms/step
- loss: 1.4742 - accuracy: 0.6812
Loss 1.4741798400878907, Accuracy 0.6812499761581421
```

You now have gone through the basic ML pipeline and trained a model to classify Fashion-MNIST data.

```
[ ]
```

Laying out notebook...

Conclusion

In this book, we provided an introduction to coding with TensorFlow 2.0. We showed how to develop with TensorFlow 1.0 and contrasted how the same code can be developed in TensorFlow 2.0.