Day 3: Hands-on AI

Part 1: Getting Started with TensorFlow

2021

Qualcomm

Part 2:
Getting Started with TensorFlow

Content

- ◆ 2.1 Building Deep Learning Model
- ◆ 2.2 Getting Started with TensorFlow
  - 2.2.1 TensorFlow Overview
  - 2.2.2 Low level API
  - 2.2.3 Middle level API
  - 2.2.4 High level API: Keras
- ◆ 2.3 Basic Knowledge of Object Detection

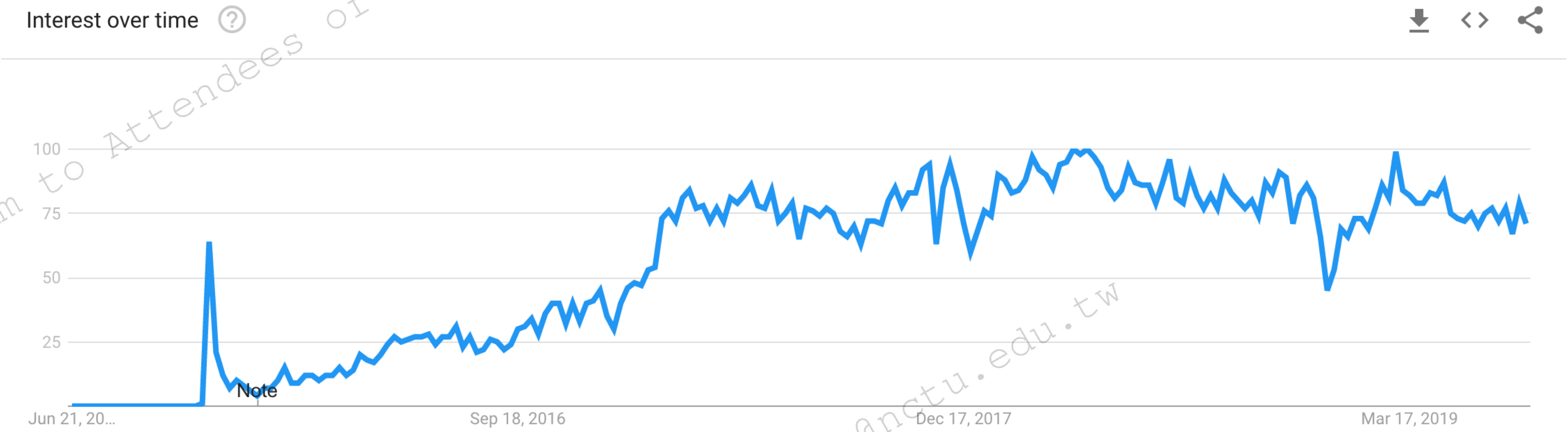
2.2 Getting Started with TensorFlow

◆ 2.2.1 TensorFlow Overview

- TensorFlow API
- Modules to build network

TensorFlow Overview

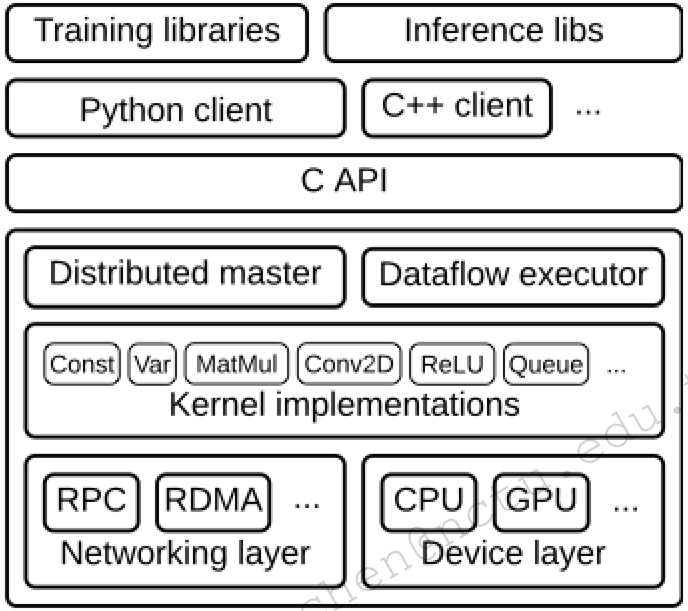
- An open source machine learning library for research and production.



From Google Trend, July 20, 2019

TensorFlow Overview

TensorFlow Architecture



https://www.tensorflow.org/guide/extend/architecture

TensorFlow APIs

High Level APIs

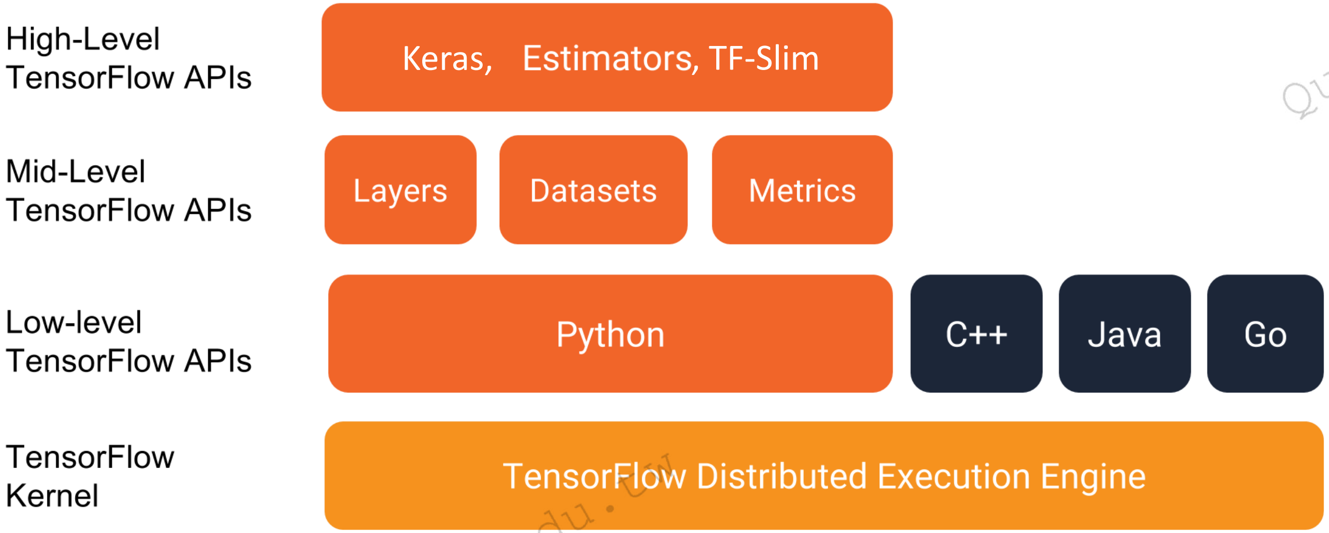
- Keras
- Eager Execution
- Importing Data
- Estimators

Middle Level APIs

- tf.layers
- tf.dataset
- tf.metrics

Low Level APIs

- Tensors
- Variables
- Graphs and Sessions
- Save and Restore


High-Level TensorFlow APIs
Keras, Estimators, TF-Slim
Mid-Level TensorFlow APIs
Layers
Datasets
Metrics
Low-level TensorFlow APIs
Python
C++
Java
Go
TensorFlow Kernel
TensorFlow Distributed Execution Engine

https://www.tensorflow.org/guide

2.2 Getting Started with TensorFlow

◆ 2.2.2 Low level API

- Graphs and Sessions
- Basic operations and Tensor types

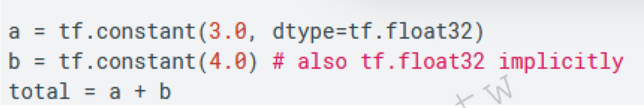
Graphs and Sessions

- TensorFlow uses a **dataflow graph** to represent your computation in terms of the dependencies between individual operations.
- A **session** is created to run parts of the graph across a set of local and remote devices.
- Two basic phases of Tensorflow computation:
  - Phase 1: assemble a graph (tf.Graph)
  - Phase 2: use a session to execute operations in the graph (tf.Session)

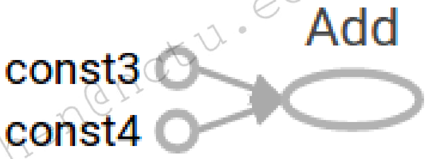
Graphs and Sessions

In a dataflow graph, the nodes represent units of computation, and the edges represent the data consumed or produced by a computation.

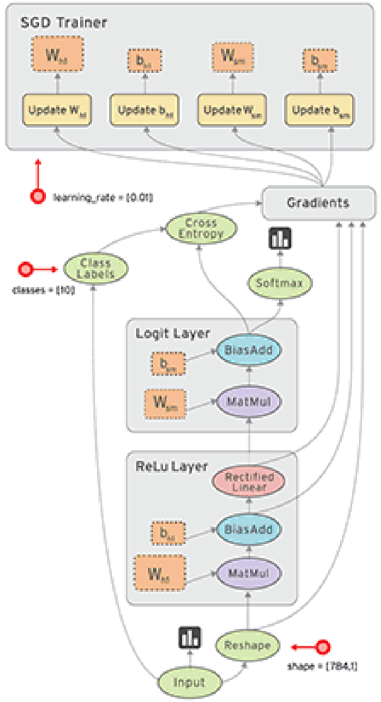
- tf.Tensor (node)
- tf.Operation (edge)



```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
```



A simple dataflow graph



A complicated dataflow graph

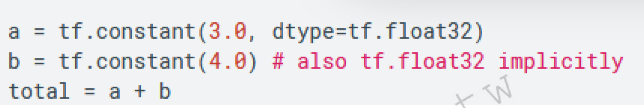
https://www.tensorflow.org/guide/graphs
https://www.tensorflow.org/guide/low_level_intro

Basic operations

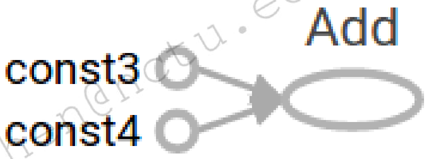
- tf.Operation represents a graph node that performs computation on tensors.
- tf.Tensor represents one of the outputs of an Operation.

- Examples of oprations:

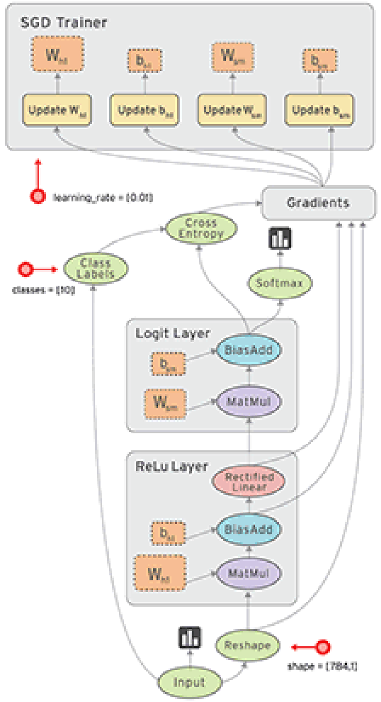
  - tf.add
  - tf.matmul,
  - tf.linalg.inv



```
a = tf.constant(3.0, dtype=tf.float32)
b = tf.constant(4.0) # also tf.float32 implicitly
total = a + b
```



A simple dataflow graph



A complicated dataflow graph

Tensor types

- tf.Tensor is **generalization** of vectors and matrices to potentially higher dimensions. It can be a vector (image label) or a 4D array (a batch of RGB image, NCHW).

- This class has two primary usage:

- 1. A Tensor can be passed as an input to another Operation. This builds a dataflow connection between operations, which enables TensorFlow to execute an entire Graph that represents a large, multi-step computation.

- 2. After the graph has been launched in a session, the value of the Tensor can be computed by passing it to tf.Session.run. t.eval() is a shortcut for calling tf.compat.v1.get_default_session().run(t).

Tensor types

- ◆ A tf.Tensor has the following properties:
  - a data type (float32, int32, or string, for example)
  - a shape
- ◆ Here are some special and common tensors:
  - tf.constant
  - tf.Variable
  - tf.placeholder
  - tf.SparseTensor

tf.Variable

- **Creating a Variable**
- tf.get_variable - Gets an existing variable with these parameters, or create a new one if it doesn’t exist.
- tf.Variable – Initilize a tf.Variable with an initial value

tf.Variable

- **Variable collections**
- Collections are named lists of tensors or other objects, such as tf.Variable instances, which provide an easy access to variables created by different parts of the program.
- By default every tf.Variable are placed in these two collections below:
  - tf.GraphKeys.GLOBAL_VARIABLES --- variables that can be shared across multiple devices(GPUs and CPU),
  - tf.GraphKeys.TRAINABLE_VARIABLES --- variables for which TensorFlow will calculate gradients.
- If you don’t to add to tf.GraphKeys.GLOBAL_VARIABLES or you need a un-trainable variable.

```
my_local = tf.get_variable("my_local", shape=(), collections=[tf.GraphKeys.LOCAL_VARIABLES])
my_non_trainable = tf.get_variable("my_non_trainable", shape=(),  trainable=False)
```

2.2 Getting Started with TensorFlow

◆ 2.2.3 Middle level API

- Optimizers
- Loss functions
- Read from dataset: tf.data, tfrecord
- Save model: Checkpoints and SavedModel
- Development Pipeline
- Tensorboard

Modules to build network in TensorFlow

- Low-level API: tf.nn
- Middle-level API: tf.layers
- High-level API: tf.contrib.slim / tf.keras

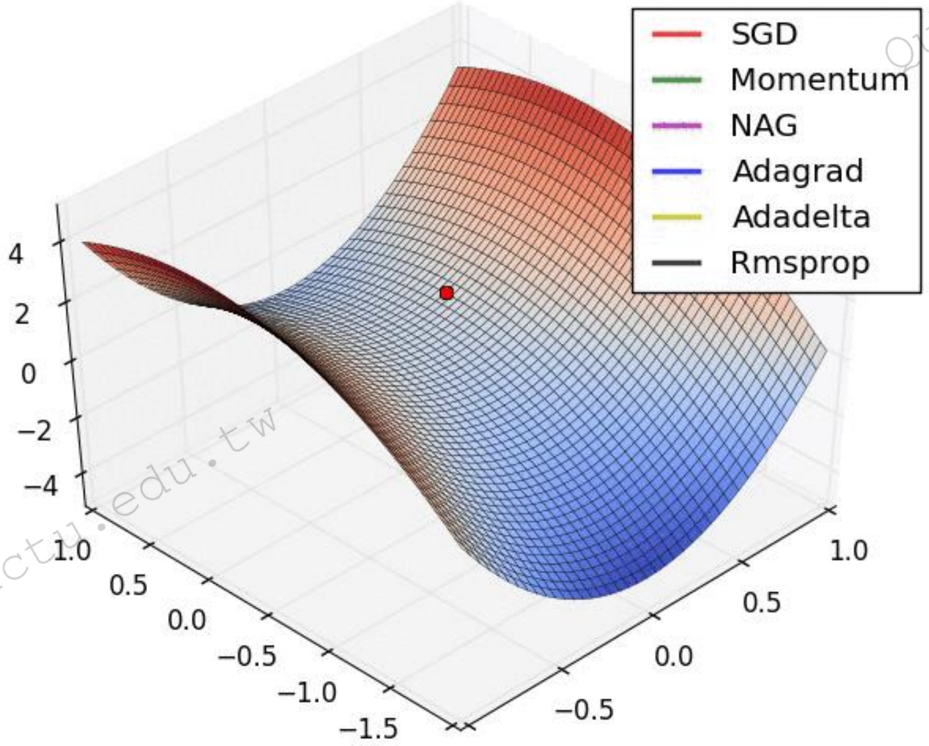
- 1. tf.nn is basic and low-level API.
- 2. tf.layers - a higher level package of tf.nn.
- 3. tf.contrib.slim - TF-Slim is a lightweight library for defining, training and evaluating complex models in TensorFlow.
- 4. tf.estimator - a high-level TensorFlow API that greatly simplifies machine learning programming.
- 5. Keras is a popular high-level API to build and train deep learning models. The computational backend supports TensorFlow, CNTK and Theano. tf.keras is TensorFlow's implementation of the Keras API specification.

https://www.tensorflow.org/guide/premade_estimators

Optimizers

- tf.train.Optimizer provides a list of optimizer like GradientDescentOptimizer(SDG), AdamOptimizer, MomentumOptimizer, etc.

```
# Create an optimizer with the desired parameters.
opt = GradientDescentOptimizer(learning_rate=0.1)
# Add Ops to the graph to minimize a cost by updating a list of variab
# "cost" is a Tensor, and the list of variables contains tf.Variable
# objects.
opt_op = opt.minimize(cost, var_list=<list of variables>)
```

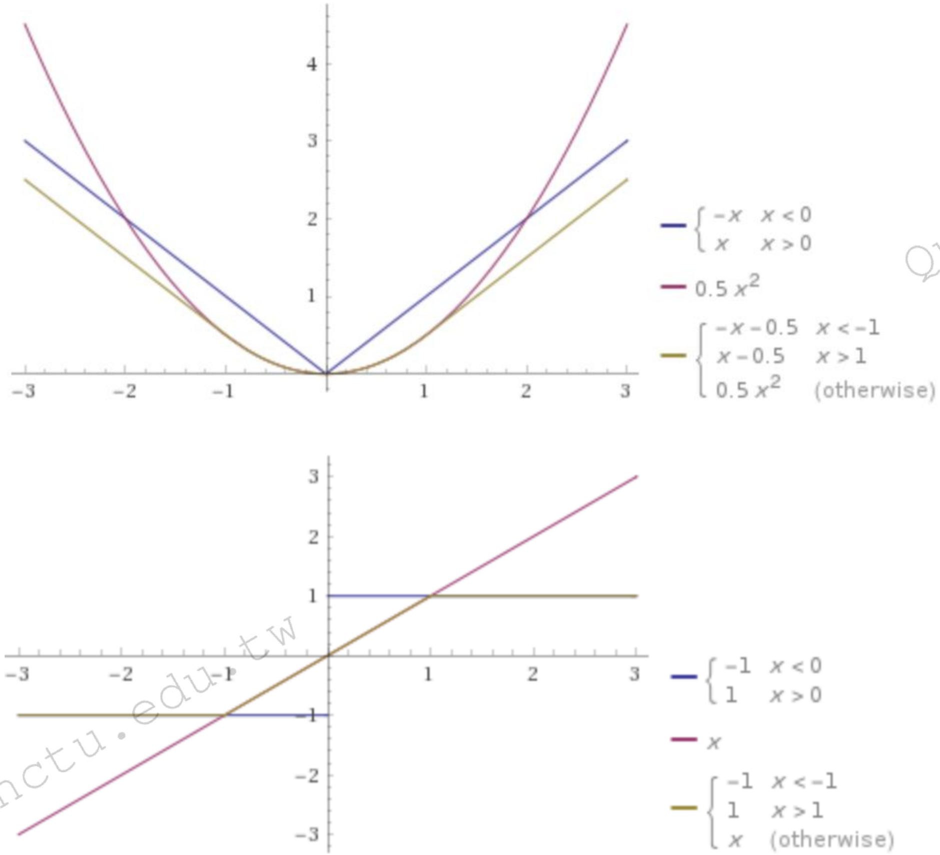


(Source: Stanford class CS231n, MIT License, Image credit: Alec Radford)

Loss functions

- tf.losses provides a lot of loss functions, such as

- - softmax_cross_entropy (for classfication)
- - hinge_loss (for classfication)
- - mean_squared_error(L2),
- - absolute_difference(L1),
- - huber_loss(smooth L1).



Read from dataset: tf.data, TFRecord

- The tf.data API provides two types of mechanisms to handle data reading.
- A tf.data.Dataset represents a sequence of elements (in tf.Tensor). E.g. a list of images and their labels.
- A tf.data.Iterator provides the main way to extract elements from a dataset.
- A tf.data.TFRecordDataset deals with reading from one or more TFRecord files.
- TFRecord is a format for storing a sequence of binary records for serialized files.

Save model: Checkpoints and SavedModel

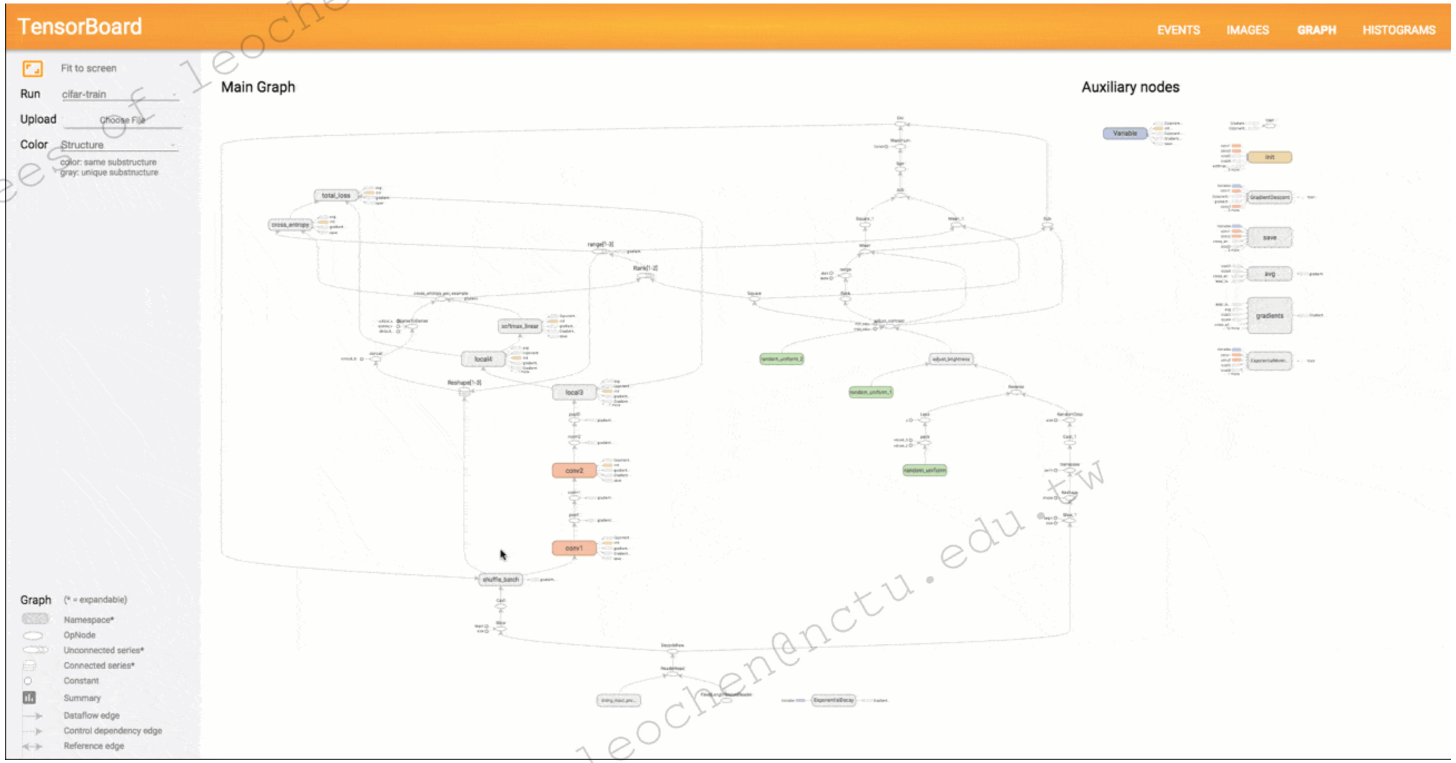
- There two types of “saving a Tensorflow model”:

- 1. Checkpoints – A snapshot of all parameters. Checkpoints don’t contain computation graph, so restoring from checkpoints need the code (Python interface) to construct the computation graph.

- 2. SavedModel – SavedModel saves a serialized description of the computation graph and the parameter values; it is independent of the source code that created the model, so it usually applies to model deployment.

Tensorboard

- Tensorboard is visualization tools to visualize your dataflow graph, draw quantitative metrics about the execution like loss and accuracy, and show additional data like images that pass through it.



Run: cifar-train

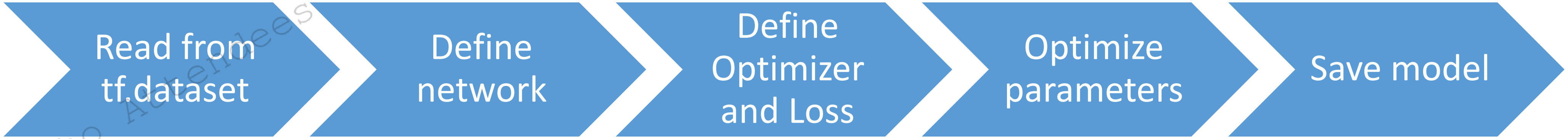
Upload: Choose File

Color: Structure

color: same substructure
gray: unique substructure

https://www.tensorflow.org/guide/graph_viz

Development Pipeline - Training



Read from tf.dataset

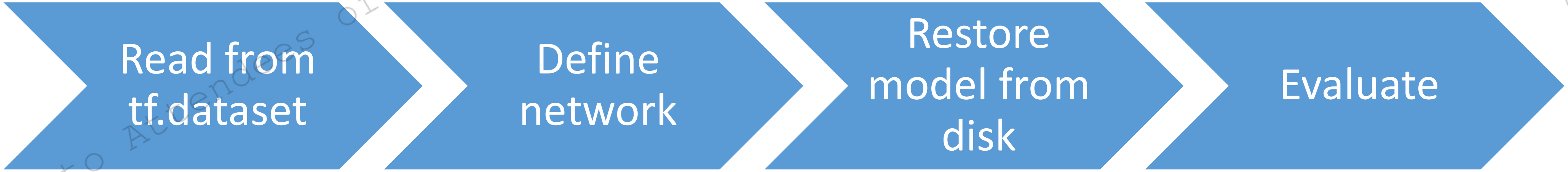
Define network

Define Optimizer and Loss

Optimize parameters

Save model

Development Pipeline - Test



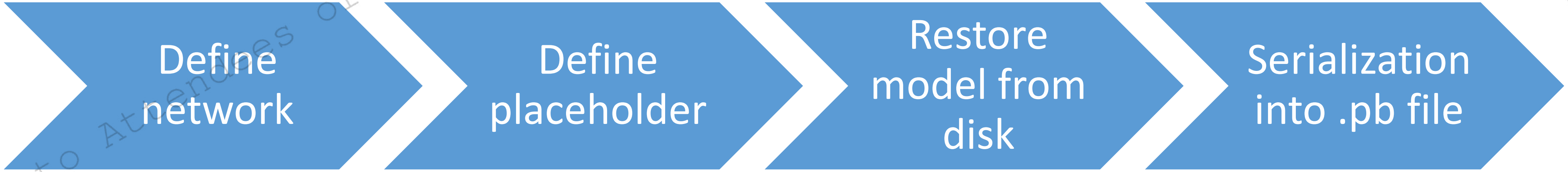
Read from tf.dataset

Define network

Restore model from disk

Evaluate

Development Pipeline - Deploy



Define network

Define placeholder

Restore model from disk

Serialization into .pb file

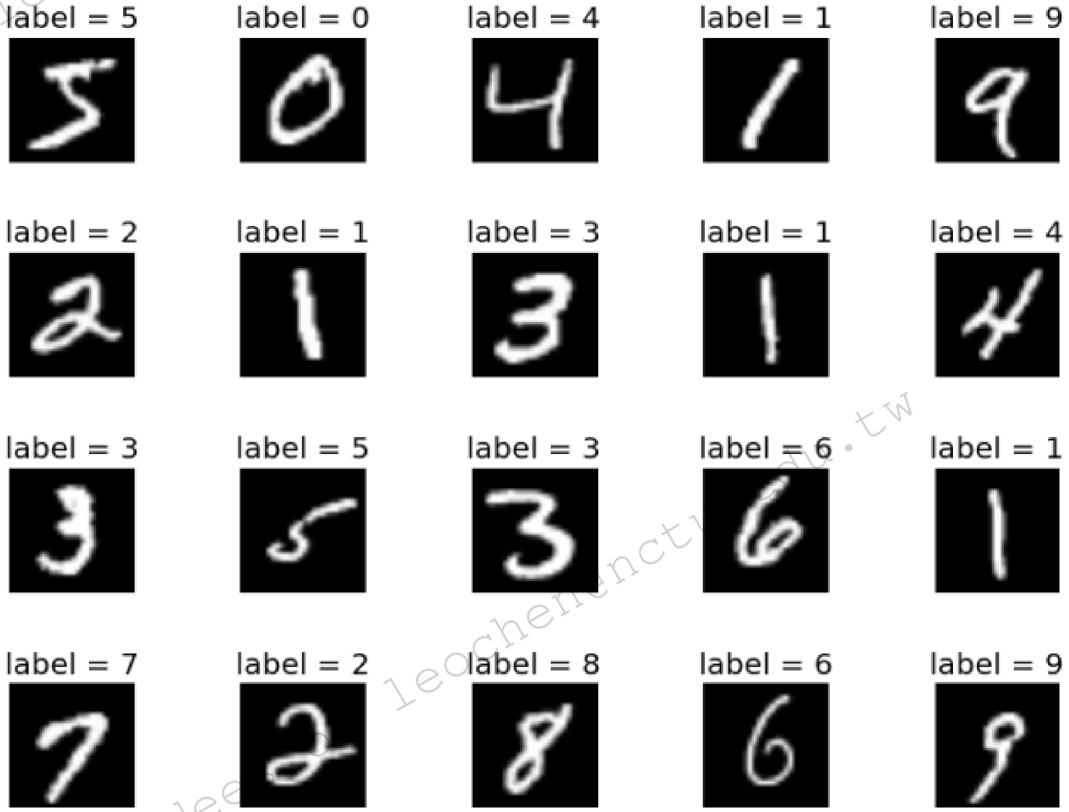
2.2 Getting Started with TensorFlow

◆ 2.2.4 High level API: Keras

- Basic knowledge of Keras
- Hands-on - TF_Keras_MNIST

Hands-on - TF_Keras_MNIST_MLP

- MNIST dataset contains 60,000 grayscale images in 10 categories. The images show hard-writing numbers at low resolution (28 by 28 pixels), image examples and all labels is as follows:



Basic knowledge of Keras

- Keras is a high-level, user-friendly neural-network library written in Python. It offers consistent & simple APIs as programing front-end, and employs popular deep learning framework as computation back-end.
- A complete life cycle of model training and evaluation includes the following steps:
  - 1. Build a network
  - 2. Compile the network with loss function, optimizer and evaluation metric provided.
  - 3. Train the model
  - 4. Evaluate it on validation dataset and output the performance metric
  - 5. Predict new images with the trained model

Please refer to Jupyter notebook for detail.

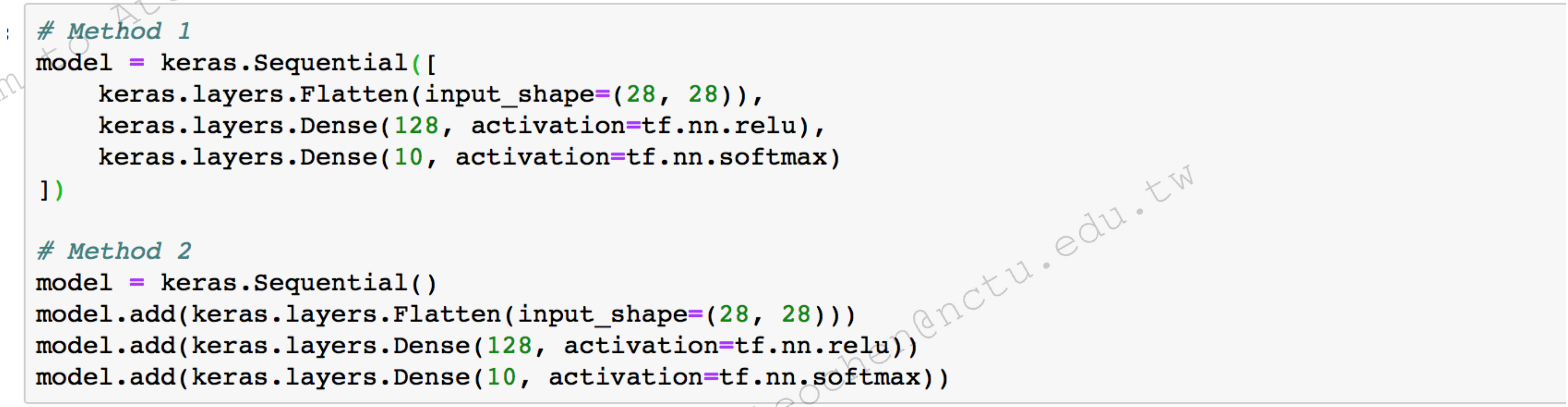
Basic knowledge of Keras

1. Build a simple network

The basic building block of a neural network is the `layer`. Each layer processes its input data and output the result to next layer.

The `Sequential` model is a linear stack of layers. There are two methods to build a sequential model.

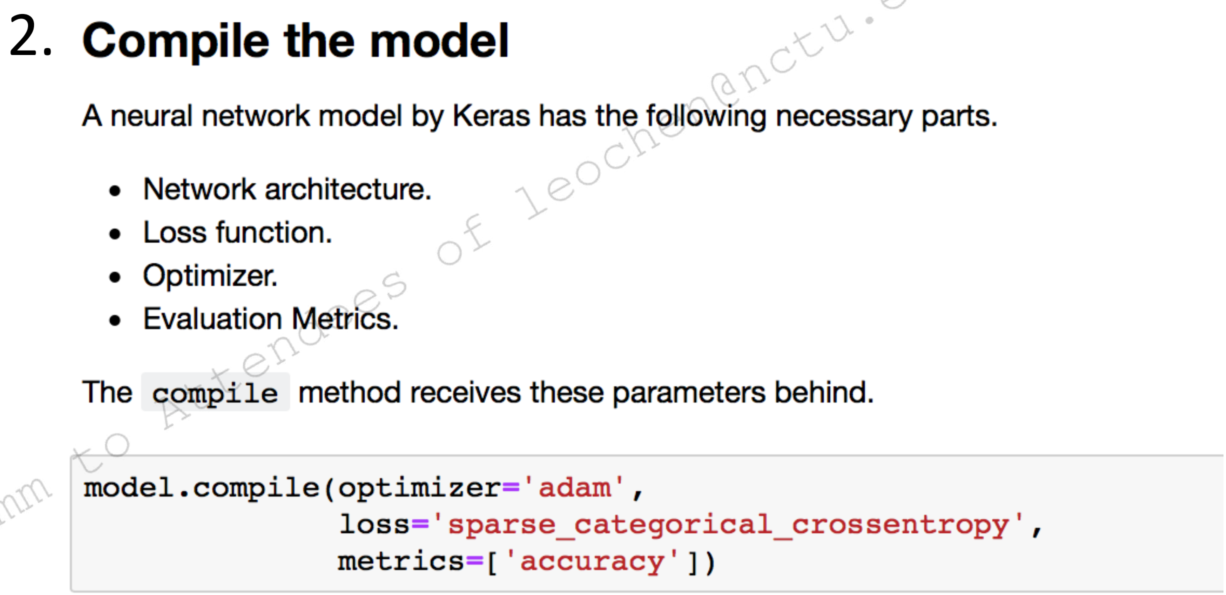
1. Pass a list of `keras.layers` to the constructor of `keras.Sequential`
2. Initilize a empty `keras.Sequential` and use `.add()` method to add layers.



```
# Method 1
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

# Method 2
model = keras.Sequential()
model.add(keras.layers.Flatten(input_shape=(28, 28)))
model.add(keras.layers.Dense(128, activation=tf.nn.relu))
model.add(keras.layers.Dense(10, activation=tf.nn.softmax))
```

Basic knowledge of Keras



2. Compile the model

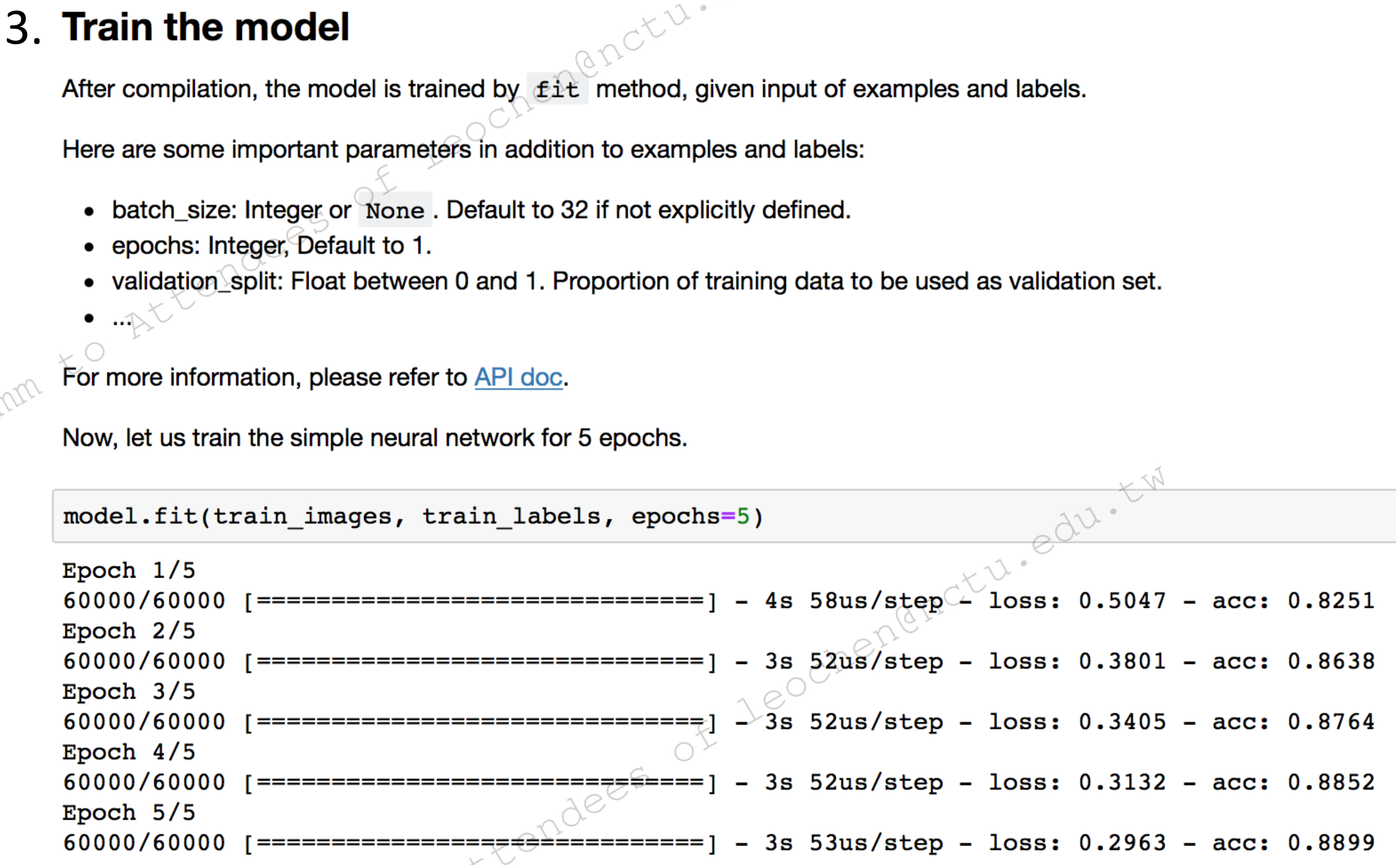
A neural network model by Keras has the following necessary parts.

- Network architecture.
- Loss function.
- Optimizer.
- Evaluation Metrics.

The `compile` method receives these parameters behind.

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

Basic knowledge of Keras



3. Train the model

After compilation, the model is trained by `fit` method, given input of examples and labels.

Here are some important parameters in addition to examples and labels:

- batch_size: Integer or `None`. Default to 32 if not explicitly defined.
- epochs: Integer, Default to 1.
- validation_split: Float between 0 and 1. Proportion of training data to be used as validation set.
- ...

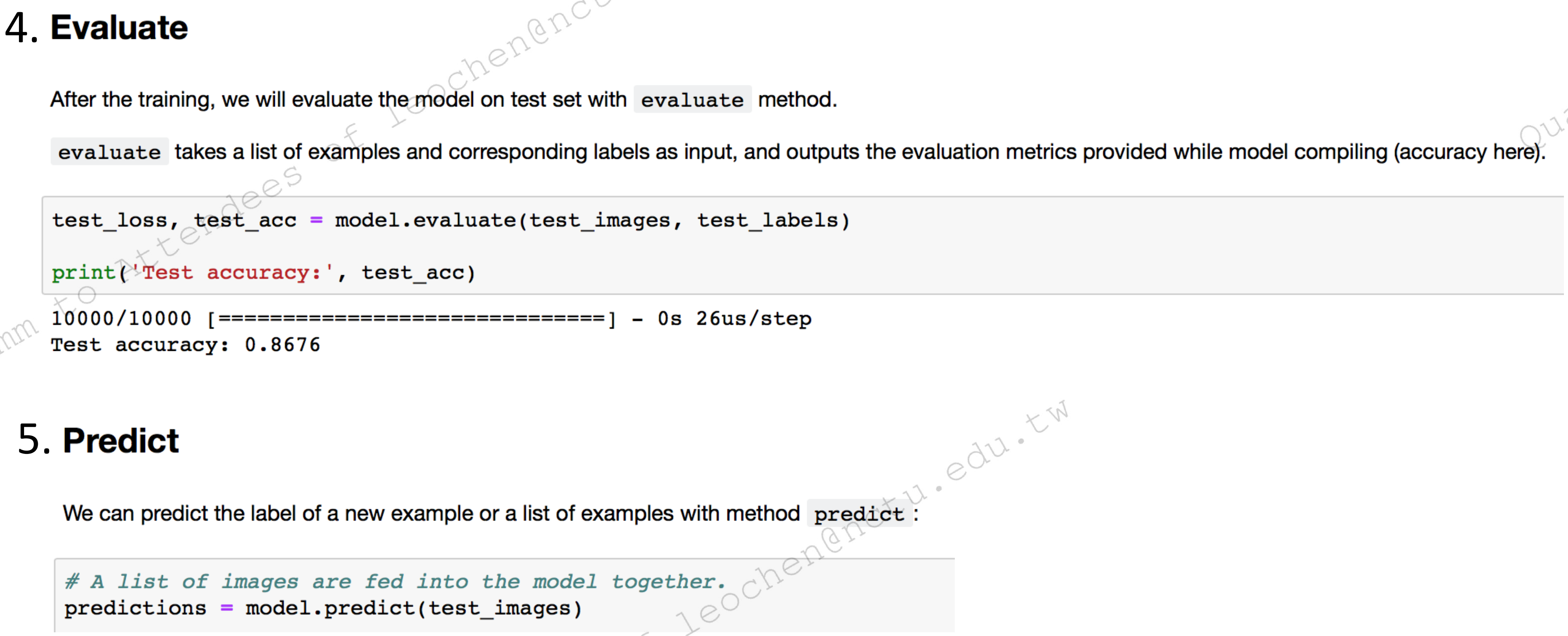
For more information, please refer to API doc.

Now, let us train the simple neural network for 5 epochs.

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

```
Epoch 1/5
60000/60000 [==============================] - 4s 58us/step - loss: 0.5047 - acc: 0.8251
Epoch 2/5
60000/60000 [==============================] - 3s 52us/step - loss: 0.3801 - acc: 0.8638
Epoch 3/5
60000/60000 [==============================] - 3s 52us/step - loss: 0.3405 - acc: 0.8764
Epoch 4/5
60000/60000 [==============================] - 3s 52us/step - loss: 0.3132 - acc: 0.8852
Epoch 5/5
60000/60000 [==============================] - 3s 53us/step - loss: 0.2963 - acc: 0.8899
```

Basic knowledge of Keras



4. Evaluate

After the training, we will evaluate the model on test set with `evaluate` method.

`evaluate` takes a list of examples and corresponding labels as input, and outputs the evaluation metrics provided while model compiling (accuracy here).

```
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

```
10000/10000 [==============================] - 0s 26us/step
Test accuracy: 0.8676
```

5. Predict

We can predict the label of a new example or a list of examples with method `predict` :

```
# A list of images are fed into the model together.
predictions = model.predict(test_images)
```

Thank You