TensorFlow2.0로 시작하는 신경망과 이미지 처리

디플러스 김영하

youngha@dplus.company

TensorFlow 2.0 소개

TensorFlow 1.x는 이제

- 옛날 코드! (Legacy code)
- 구글은 보안과 관련된 업데이트만 제공할 예정입니다.
- 앞으로 몇 년 더 유지될지는...

TensorFlow 2.0은

- 이전과 마찬가지로 구글이 만든 오픈 소스 프레임워크!
- 머신러닝과 딥러닝을 위한 라이브러리!
- Apache 라이선스!
- TensorFlow 1.x에 존재한 중복된 API 제거, 빠른 프로토타입 및 쉬운 디버깅이 가능!

TensorFlow 2.0은

- 기본 실행 모드가 deferred execution이 아니라, eager execution입니다.
- 핵심 기능의 일부로 Keras를 채택했습니다!
- Keras를 위해 개발하고 최적화해서 당연히 Keras의 모든 기능을 사용 가능
- 딥러닝 모델 (CNN, RNN, LSTM 등)을 사용할 경우, **tf.keras** 네임스페이스를 사용

TensorFlow 2.0은

- 하위 호환성을 제공합니다.
  - ✓ tf.compat.v1 모듈 제공 (tf.contrib 미포함)
  - ✓ tf_upgrade_v2 라는 변환 스크립트를 제공
- tensorflow.js v1.0 을 제공합니다.
- 분산 데이터를 위한 TensorFlow Federated를 제공합니다.
- ragged tensors를 지원합니다.
  - ✓ tf.ragged.constant([[3, 1, 4, 1], [], [5, 9, 2], [6], []])
- 확률적 추론 및 통계분석을 위한 TensorFlow Probability를 제공합니다.

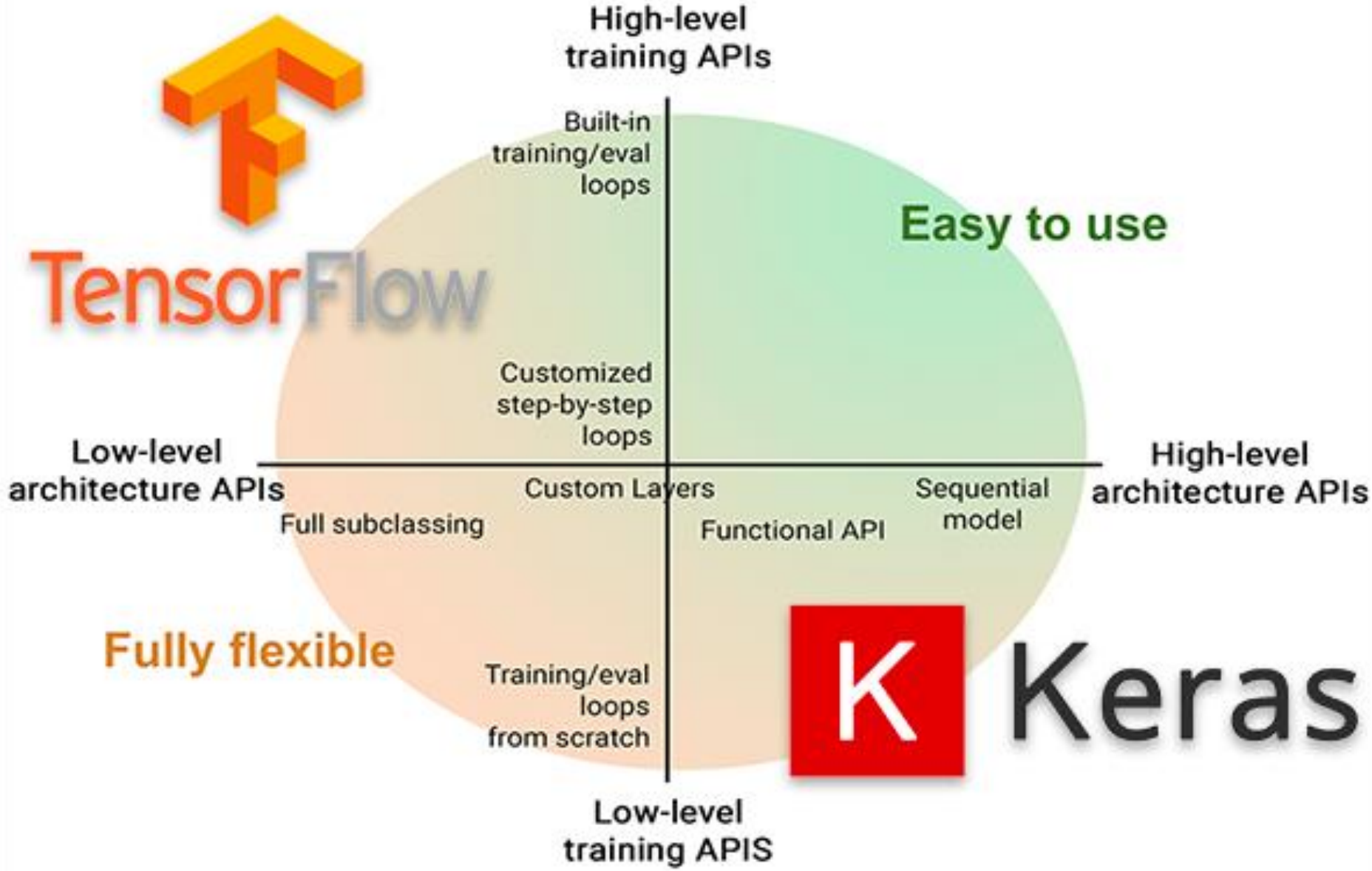
```
(venv) D:₩workspace₩tensorflow2>tf_upgrade_v2
usage: tf_upgrade_v2 [-h] [--infile INPUT_FILE] [--outfile OUTPUT_FILE]
                     [--intree INPUT_TREE] [--outtree OUTPUT_TREE]
                     [--copyotherfiles COPY_OTHER_FILES] [--inplace]
                     [--reportfile REPORT_FILENAME] [--mode {DEFAULT,SAFETY}]
                     [--print_all]
Convert a TensorFlow Python file from 1.x to 2.0

Simple usage:
  tf_upgrade_v2.py --infile foo.py --outfile bar.py
  tf_upgrade_v2.py --infile foo.ipynb --outfile bar.ipynb
  tf_upgrade_v2.py --intree ~/code/old --outtree ~/code/new

optional arguments:
  -h, --help            show this help message and exit
  --infile INPUT_FILE   If converting a single file, the name of the file to convert
  --outfile OUTPUT_FILE
                        If converting a single file, the output filename.
  --intree INPUT_TREE   If converting a whole tree of files, the directory to
                        read from (relative or absolute).
  --outtree OUTPUT_TREE
                        If converting a whole tree of files, the output
                        directory (relative or absolute).
                        behavior continues to work.
  --print_all           Print full log to stdout instead of just printing errors
```

참조 : https://www.tensorflow.org/guide/upgrade

TensorFlow 2.0은

- @tf.function decorator를 지원합니다.
  - ✓ 파이썬 decorato입니다.
  - ✓ 그래프를 정의하고 session 실행을 합니다.
  - ✓ TensorFlow 1.x의 tf.Session()과 유사합니다.
- AutoGraph를 지원합니다.
  - ✓ 파이썬 소스코드를 그래프로 변환합니다.
  - ✓ @tf.function 사용시 자동으로 적용합니다.
- Automatic differentiation and GradientTape

근데 TensorFlow 2.0과 keras는 무슨 관계야?
keras로 개발하는데 TensorFlow 2.0?
신경망을 학습시키는데 keras 패키지를 사용해야하나요
아니면 tf.keras를 사용해야하나요?
어떤 TensorFlow 2.0 기능에 관심을 가져야 하나요?


High-level training APIs
Built-in training/eval loops
Easy to use
TensorFlow
Customized step-by-step loops
Low-level architecture APIs
High-level architecture APIs
Custom Layers
Sequential model
Full subclassing
Functional API
Fully flexible
Training/eval loops from scratch
K
Keras
Low-level training APIS

TensorFlow과 Keras와의 역사를 이해하면 됩니다

TensorFlow와 Keras의 역사

Francois Chollet
구글 AI 개발자/연구원

Francois Chollet 가 말씀하시길

1. Keras와 tf.keras가 이제 같습니다.
2. 이제 앞으로는 멀티 백엔드 지원 안 합니다.
3. 앞으로 TensorFlow 2.0과 tf.keras를 사용하세요.
4. Keras는 버그 정도만 수정할 예정!

2015년 3월 27일 Keras 첫 공개 (백엔드 : Theano)

Torch, Theano, Caffe와 같은 딥러닝 라이브러리가 있었지만, 어셈블리나 C++을 사용해야 했고, 시간이 많이 소요되고 비효율적!?

Keras는 사용하기도 쉽고, 작업 능률도 좋았습니다.
그러나, 이런 백엔드 (backend)가 필요했습니다.

2015년 11월 9일 TensorFlow 첫 공개

천천히, 조금씩 Keras는 백엔드로 TensorFlow를 지원하기 시작합니다

2016년 9월 20일 Keras 1.1.0 공개

기본 백엔드가 TensorFlow로 변경됩니다.

서브 모듈에 tf.keras가 포함됩니다.
즉, TensorFlow와 Keras와의 통합이 시작됩니다.
이제 Keras는 tf.keras는 별도의 패키지!?

2018년 8월 9일 TensorFlow 1.10.0 공개

2019년 6월 구글이 TensorFlow 2.0 개발을 알림

Keras가 TensorFlow의 공식 고수준 API라고 선언합니다!?

2019년 9월 18일 Keras 2.3.0 공개

TensorFlow 1.x와 TensorFlow 2.0 비교

TensorFlow 1.x (legacy)

```
with tf.Session() as session:
    session.run(tf.global_variables_initializer())
    session.run(tf.tables_initializer())
    model.fit(
        X_train,
        y_train,
        validation_data=(
            X_valid,
            y_valid
        ),
        epochs=10,
        batch_size=64
    )
```

TensorFlow 2.0

```
model.fit(
    X_train,
    y_train,
    validation_data=(
        X_valid,
        y_valid
    ),
    epochs=10,
    batch_size=64
)
```

참조 : https://medium.com/coding-blocks/eager-execution-in-tensorflow-a-more-pythonic-way-of-building-models-e461810618c8 (자세한 Eager Execution)

참조 : https://github.com/sayakpaul/TF-2.0-Hacks/tree/master/Speed%20comparison%20between%20TF%201.x%20and%20TF%202.0 (Eager Execution과 Session Execution 차이)

TensorFlow Code

```
import tensorflow as tf

with tf.variable_scope('input'):
   X = tf.placeholder(tf.float32, shape=(None, 10), name="X")

with tf.variable_scope('layer_1'):
   weights = tf.get_variable("weights1", shape=[10, 50], initializer=tf.contrib.layers.xavier_initializer())
   biases = tf.get_variable(name="biases1", shape=[50], initializer=tf.zeros_initializer())
   layer_1_output = tf.nn.relu(tf.matmul(X, weights) + biases)

with tf.variable_scope('layer_2'):
   weights = tf.get_variable("weights2", shape=[50, 100], initializer=tf.contrib.layers.xavier_initializer())
   biases = tf.get_variable(name="biases2", shape=[100], initializer=tf.zeros_initializer())
   layer_2_output = tf.nn.relu(tf.matmul(layer_1_output, weights) + biases)

with tf.variable_scope('layer_3'):
   weights = tf.get_variable("weights3", shape=[100, 50], initializer=tf.contrib.layers.xavier_initializer())
   biases = tf.get_variable(name="biases3", shape=[50], initializer=tf.zeros_initializer())
   layer_3_output = tf.nn.relu(tf.matmul(layer_2_output, weights) + biases)

with tf.variable_scope('output'):
   weights = tf.get_variable("weights4", shape=[50, 1], initializer=tf.contrib.layers.xavier_initializer())
   biases = tf.get_variable(name="biases4", shape=[1], initializer=tf.zeros_initializer())
   prediction = tf.matmul(layer_3_output, weights) + biases

with tf.variable_scope('cost'):
   Y = tf.placeholder(tf.float32, shape=(None, 1), name="Y")
   cost = tf.reduce_mean(tf.squared_difference(prediction, Y))

with tf.variable_scope('train'):
   optimizer = tf.train.AdamOptimizer(0.05).minimize(cost)
```

TensorFlow

Equivalent Keras Code

```
from keras.models import Sequential
from keras.layers import *

model = Sequential()
model.add(Dense(50, input_dim=10, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(50, activation='relu'))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

Custom layer or loss functions

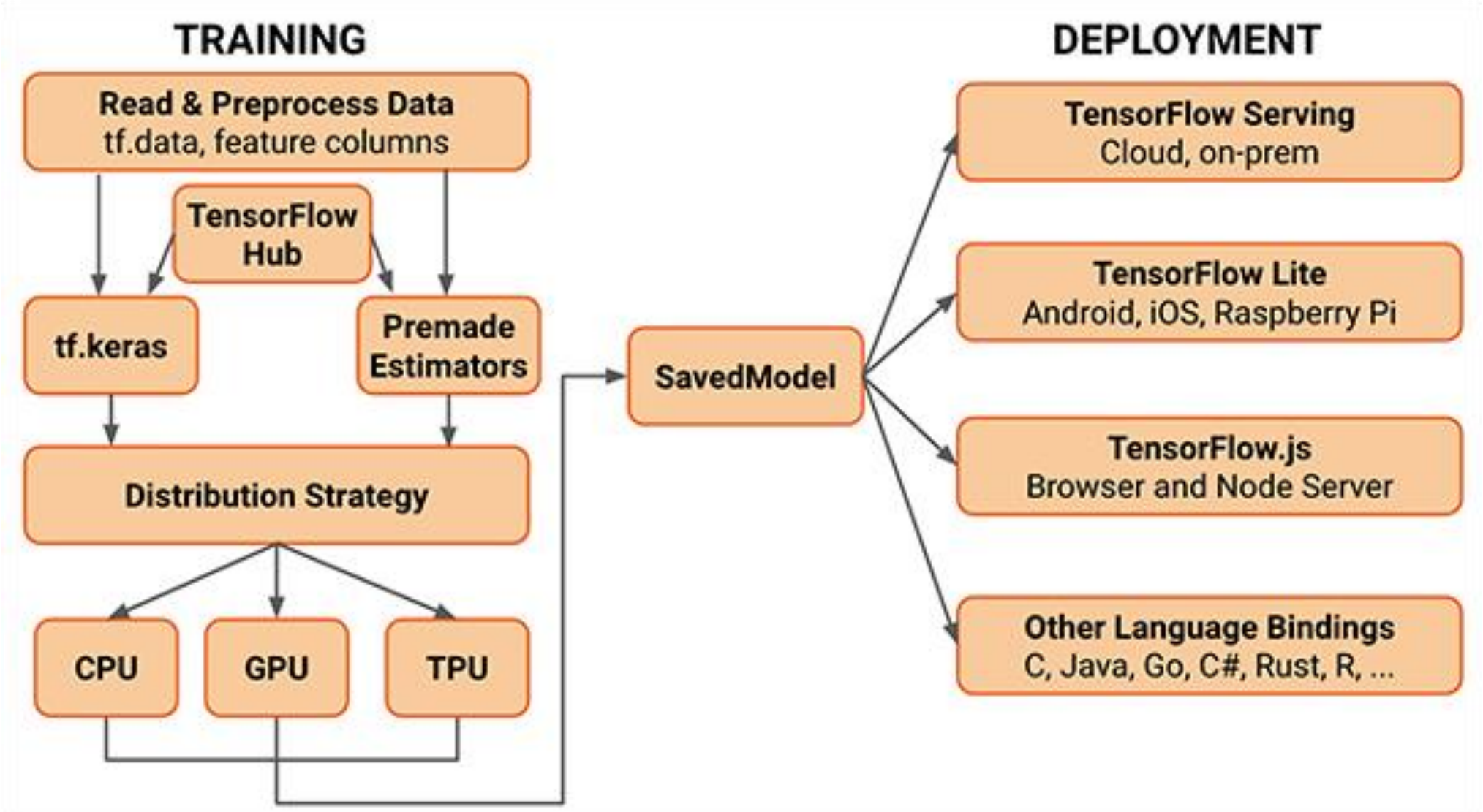
François Chollet 
@fchollet

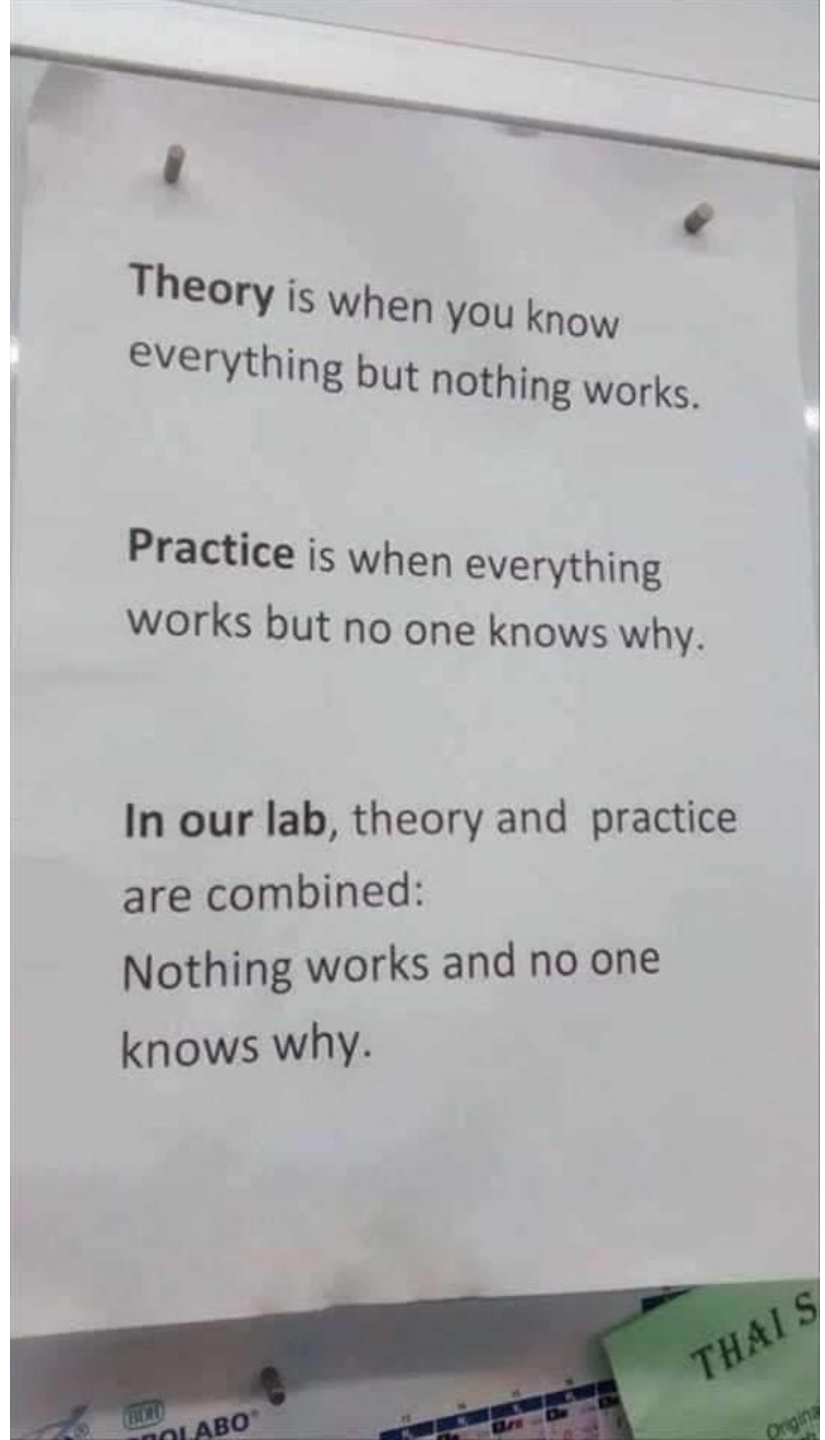
One of the big changes in TF 2.0 is how easy it is to write custom training loops for your models: tensorflow.org/beta/guide/ker...

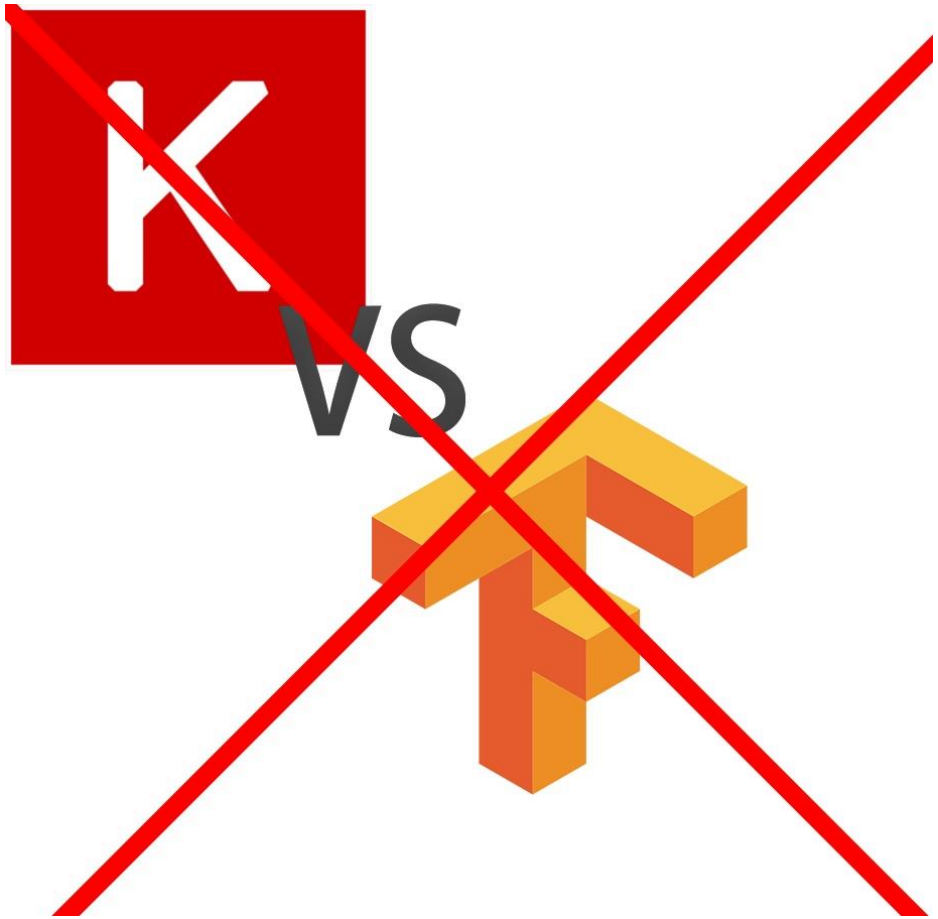
트윗 번역하기

```
# Iterate over the batches of the dataset.
for step, (x_batch_train, y_batch_train) in enumerate(train_dataset):

  # Open a GradientTape to record the operations run
  # during the forward pass, which enables autodifferentiation.
  with tf.GradientTape() as tape:

    # Run the forward pass of the layer.
    # The operations that the layer applies
    # to its inputs are going to be recorded
    # on the GradientTape.
    logits = model(x_batch_train)  # Logits for this minibatch

    # Compute the loss value for this minibatch.
    loss_value = loss_fn(y_batch_train, logits)

  # Use the gradient tape to automatically retrieve
  # the gradients of the trainable variables with respect to the loss.
  grads = tape.gradient(loss_value, model.trainable_weights)

  # Run one step of gradient descent by updating
  # the value of the variables to minimize the loss.
  optimizer.apply_gradients(zip(grads, model.trainable_weights))
```

TensorFlow 2.0 Eco System

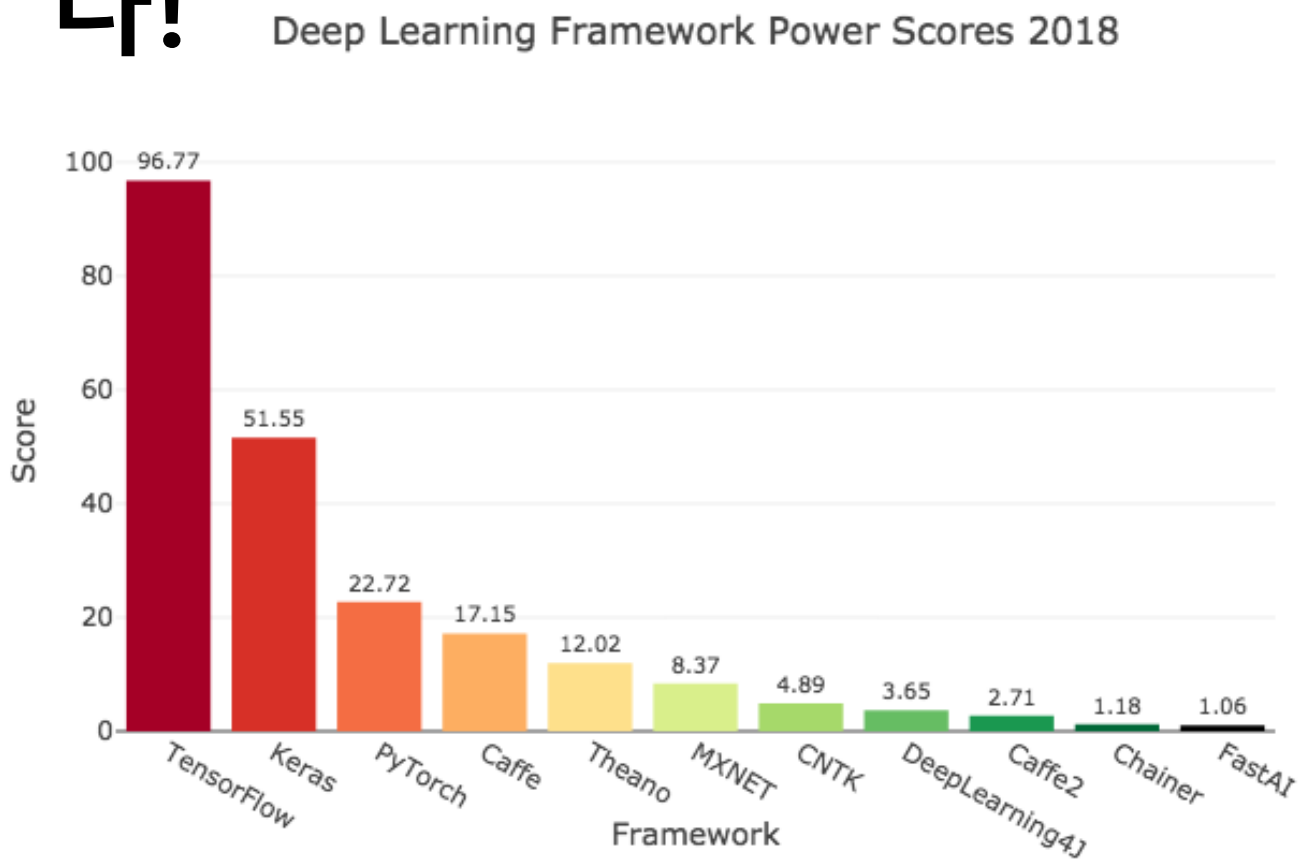



Theory is when you know
everything but nothing works.
Practice is when everything
works but no one knows why.
In our lab, theory and practice
are combined:
Nothing works and no one
knows why.



이제 케라스가 좋냐 텐서플로가 좋냐의 문제가 아닙니다!

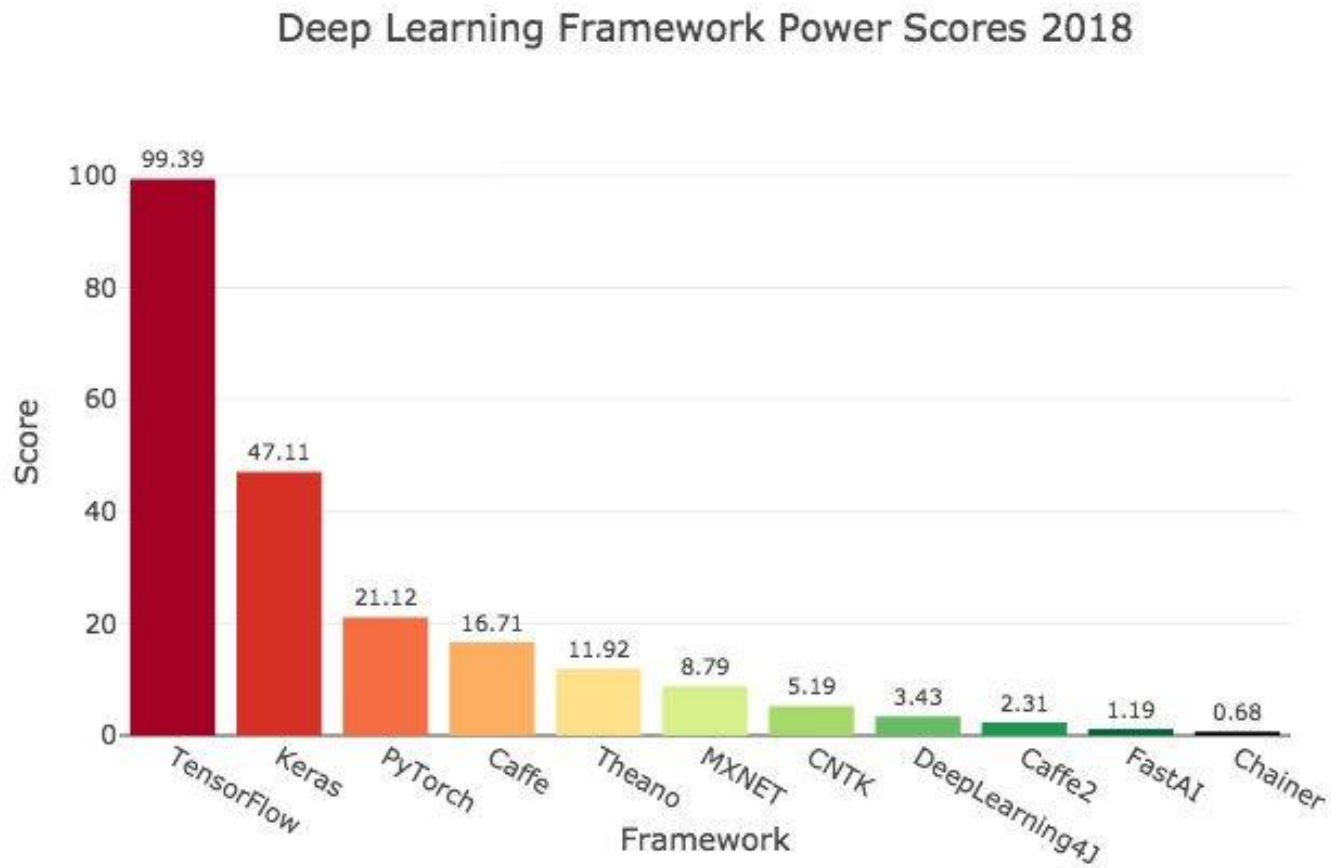
케라스는 많은 산업분야와 연구에서 사용합니다!



11개 자료와 7개 분류를 기반으로 Jeff Hale이 계산한 딥러닝 프레임워크 순위

https://keras.io/why-use-keras/

케라스는 많은 산업분야와 연구에서 사용합니다!



2018년 9월 20일에 다시 조사했어요? 그래도 역시 케라스가 1위입니다?

https://keras.io/why-use-keras/

다음 중 어떤 것이 케라스를 잘 설명한 것일까요?

- 케라스는 텐서플로와 통합되어 있습니다.
  텐서플로에서 쉽게 케라스를 사용할 수 있다는 뜻이며,
  최고의 결과를 얻을 수 있습니다.

- 케라스는 다른 백엔드 라이브러리가 없어도
  케라스 자체만으로도 자신의 역할을 할 수 있습니다.

- 프랑소와 숄레(François Chollet)가 만든 오픈소스 프로젝트입니다.

다음 중 어떤 것이 케라스를 잘 설명한 것일까요?

- **케라스는 텐서플로와 통합되어 있습니다.**
  텐서플로에서 쉽게 케라스를 사용할 수 있다는 뜻이며,
  최고의 결과를 얻을 수 있습니다.

- 케라스는 다른 백엔드 라이브러리가 없어도
  케라스 자체만으로도 자신의 역할을 할 수 있습니다.

- **프랑소와 숄레(François Chollet)가 만든 오픈소스 프로젝트입니다.**

중고나라에 물건을 팔려고 합니다.

이 때, 제품의 상표 및 정보를 자동으로 인식하였으면 합니다.

딥러닝을 활용해야 할까요?

중고나라에 물건을 팔려고 합니다.

이 때, 제품의 상표 및 정보를 자동으로 인식하였으면 합니다.

딥러닝을 활용해야 할까요?

이미지로써 비정형이라 딥러닝을 사용해보고자 합니다.

신경망이 이미지 처리에 좋을 것 같습니다.

3가지 API 방법

- Sequential Model

- 아주 쉽습니다
- 1개 입력과 1개 출력으로 계속 처리합니다
- 70%의 경우에 적용이 가능합니다

- Functional API

- 레고 블록하는 것과 같습니다
- 다중 입력, 다중 출력 등으로 처리합니다
- 95%의 경우에 적용이 가능합니다

- Model 서브클래스

- 가장 유연하게 사용할 수 있습니다. 그러나, 어렵고 복잡스럽습니다.
- 잠재적으로 오류를 유발시킬 수 있습니다

Sequential API

```
from keras.models import Sequential
from keras.layers import Dense

model = keras.Sequential()
model.add(Dense(20, input_shape=(10, ), activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

Functional API

```
from keras.models import Sequential
from keras.layers import Dense

input = keras.Input(shape=(10, ))
x = Dense(20, activation='relu')(x)
x = Dense(20, activation='relu')(x)
outputs = Dense(10, activation='softman')(x)

model = Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

Model 서브클래스

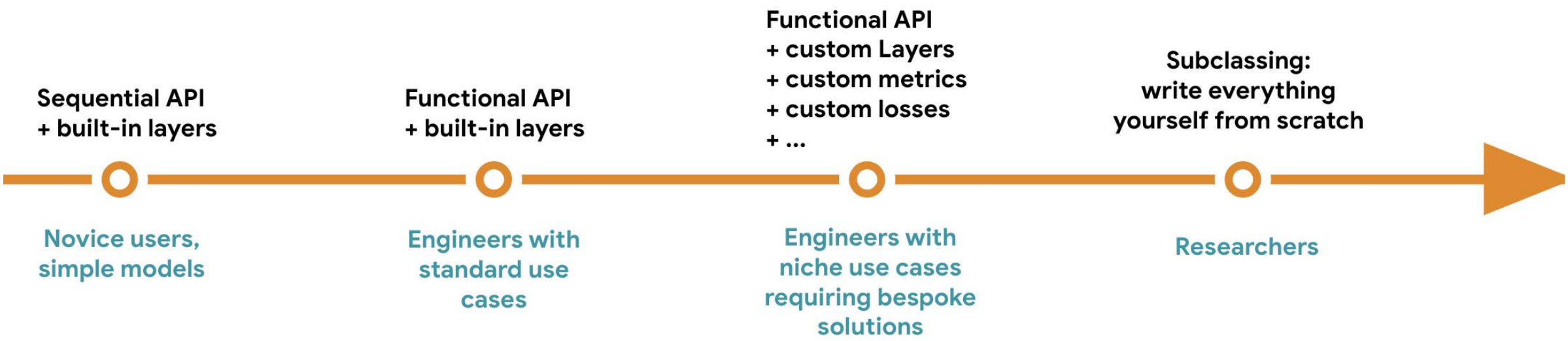
```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```

어떤 것을 사용하느냐
는
개인의 선택입니다!

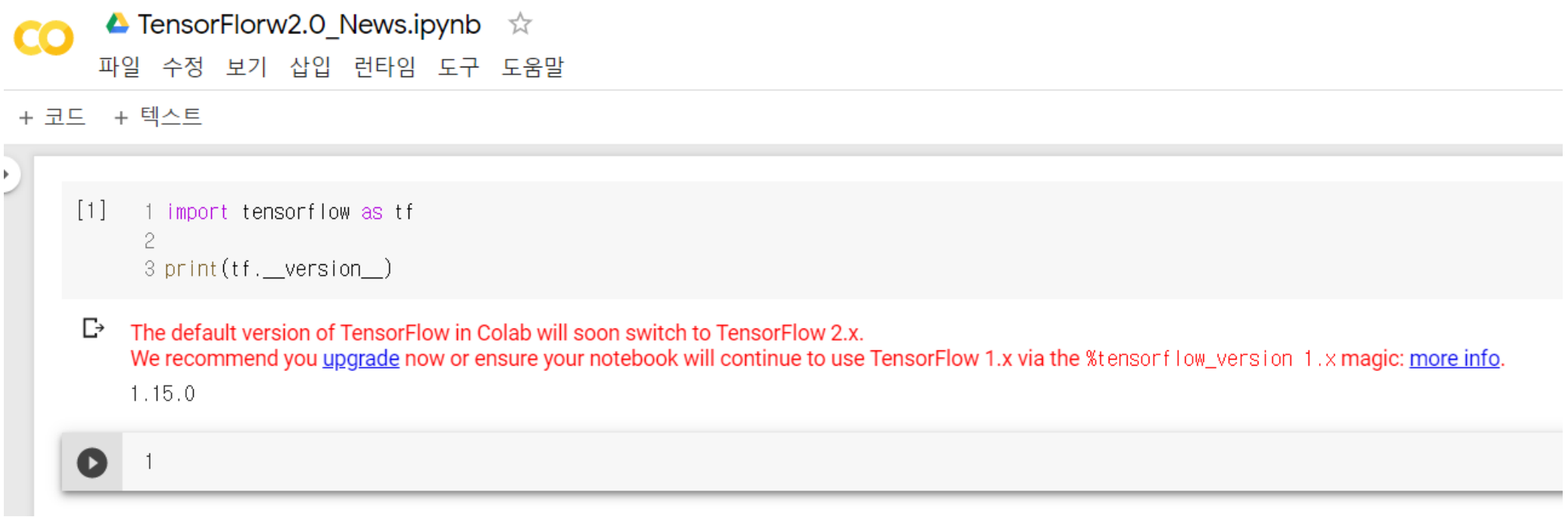
그래도 이정도의 가이드는 있습니다.

Model building: from **simple** to **arbitrarily flexible**

Progressive disclosure of complexity



아쉽게도 2019년 12월 17일 화요일 현재
Google Colab에서는 1.15.0 버전입니다.



설치법 1

```
1 !pip install grpcio>=1.24.3
2 !pip install google-auth~=1.4.0
3 !pip install tensorflow==2.0.0
4
5 # GPU version
6 # !pip install -q tensorflow-gpu==2.0.0
```

```
Collecting google-auth~=1.4.0
  Downloading https://files.pythonhosted.org/packages/56/80/369a47c28ce7d9be6a6973338133d073864d8efbb62747e414c34a3a5f4f/google_auth-1.4.2-py2.py3-none-any.whl (64kB)
     |████████████████████████████████| 71kB 4.9MB/s
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.6/dist-packages (from google-auth~=1.4.0) (0.2.7)
Requirement already satisfied: cachetools>=2.0.0 in /usr/local/lib/python3.6/dist-packages (from google-auth~=1.4.0) (3.1.1)
Requirement already satisfied: rsa>=3.1.4 in /usr/local/lib/python3.6/dist-packages (from google-auth~=1.4.0) (4.0)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.6/dist-packages (from google-auth~=1.4.0) (1.12.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /usr/local/lib/python3.6/dist-packages (from pyasn1-modules>=0.2.1->google-auth~=1.4.0) (0.4.7)
ERROR: tensorboard 2.0.1 has requirement google-auth<2,>=1.6.3, but you'll have google-auth 1.4.2 which is incompatible.
ERROR: tensorboard 2.0.1 has requirement grpcio>=1.24.3, but you'll have grpcio 1.15.0 which is incompatible.
Installing collected packages: google-auth
  Found existing installation: google-auth 1.6.3
    Uninstalling google-auth-1.6.3:
      Successfully uninstalled google-auth-1.6.3
Successfully installed google-auth-1.4.2
WARNING: The following packages were previously imported in this runtime:
  [google]
You must restart the runtime in order to use newly installed versions.
```

RESTART RUNTIME

설치법 2

```
1 !pip install --upgrade tensorflow
2
```

```
1 import tensorflow as tf
2
3 print(tf.__version__)
```

```
2.0.0
```

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

(x_train, y_train),(x_test, y_test) = mnist.load_data()

x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential(
[
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])

model.summary()

model.compile(
    optimizer='adam’,
    loss='sparse_categorical_crossentropy’,
    metrics=['accuracy'
])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

케라스와 MNIST 데이터셋

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

train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))

train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(tf.keras.layers.MaxPooling2D((2, 2)))
model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

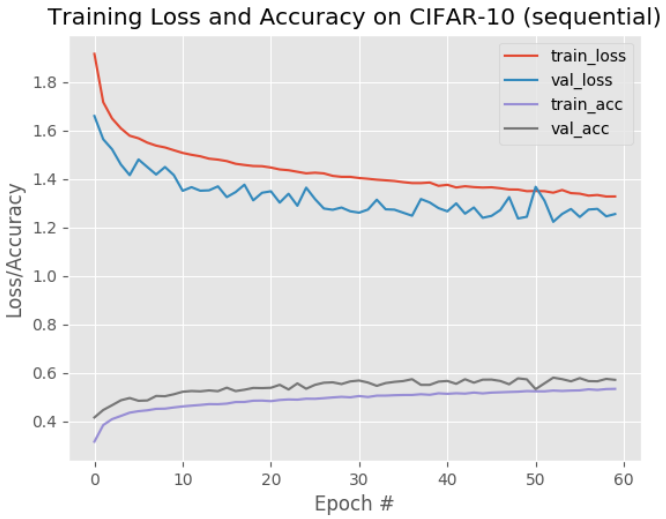
```
model.summary()
```

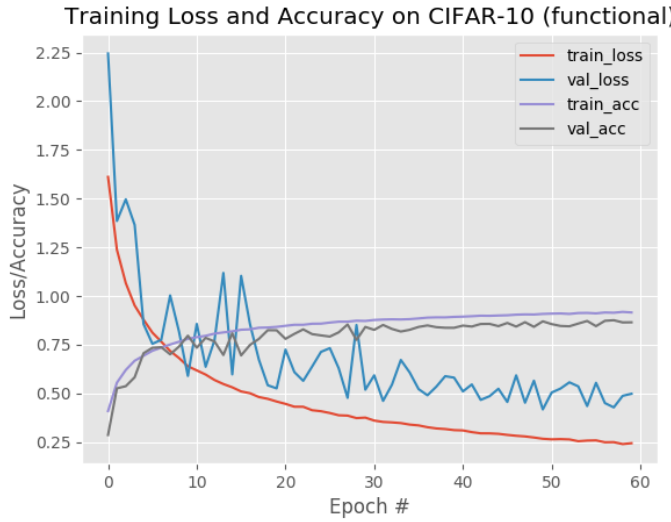
케라스와 MNIST 데이터셋 CNN 방식

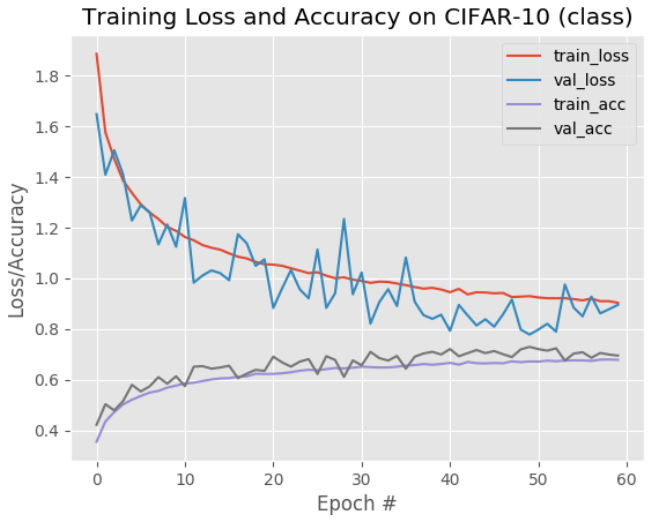
```
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=1)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(test_acc)

test_image = np.expand_dims(test_images[300],axis = 0)
plt.imshow(test_image.reshape(28,28))
plt.show()

result = model.predict(test_image)
print("result:", result)
print("result.argmax():", result.argmax())
```

케라스와 MNIST 데이터셋 CNN 방식 (계속)

3가지 방식에 대한 CIFAR-10 데이터셋에 대한 결과 비교







감사합니다!