Keras 한 층 더 멋지게 활용하기

19.10.16

Junwon Hwang

Junwon Hwang

Computer Science of Sungkyunkwan Univ.

Research Engineer in ESTsoft.

Admin of KerasKorea

E-mail: nuclear1221@gmail.com

Facebook: https://www.facebook.com/nuxlearHwang

GitHub: https://github.com/nuxlear

Intro

Tensorflow 2.0 + Keras

Keras를 쓰는 사람이 많아진다

Keras로 무언가 구현하려 할 때
어려움을 겪는 경우가 많다

Keras에서 **xxx**는 어떻게 구현하나요?

이런 것도 **Keras**로 되나요?

XXX랑 **Keras**랑 같이 쓸 수 있나요?

유용한 기능들을 소개하고 구현 방법을 공유하는
자료를 만들고 싶다

데이터를 멋지게 준비하자

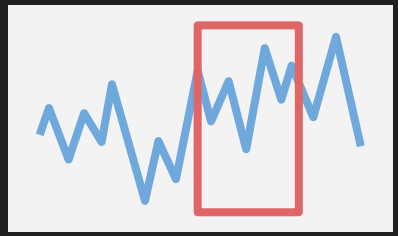
model.fit()은 편리하다

```
model.fit(x_train, y_train, batch_size)
```

별다른 코드 없이 데이터를 빠르게 넘겨줄 수 있다

만약 이런 상황이면?

데이터가 너무 커서
한 번에 불러올 수 없거나...

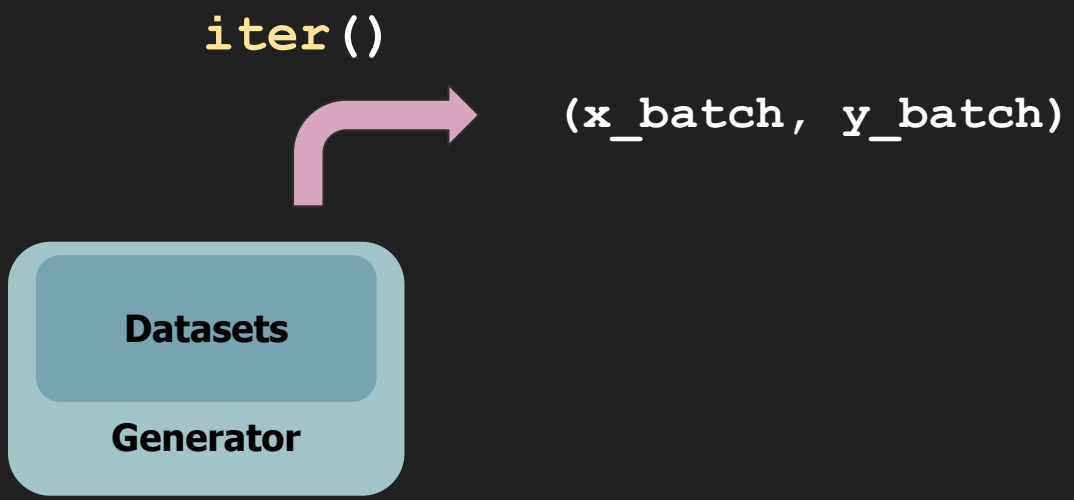


매번 임의의 구간을
샘플로 뽑아 사용한다거나...

데이터를 그대로 넣어주기 어려울 때가 많다

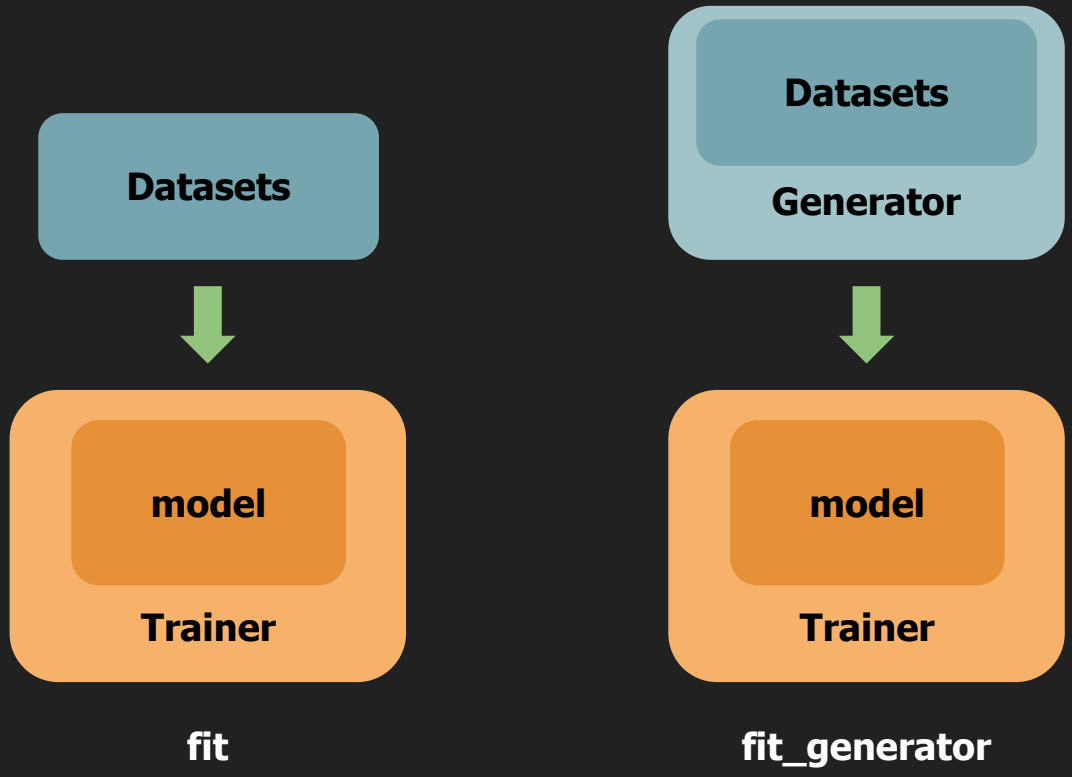
그러면? 필요할 때 가져오자!

generator는
데이터를 생성하는 iterator다



model trainer가 요청할 때 **batch**를 넘겨준다

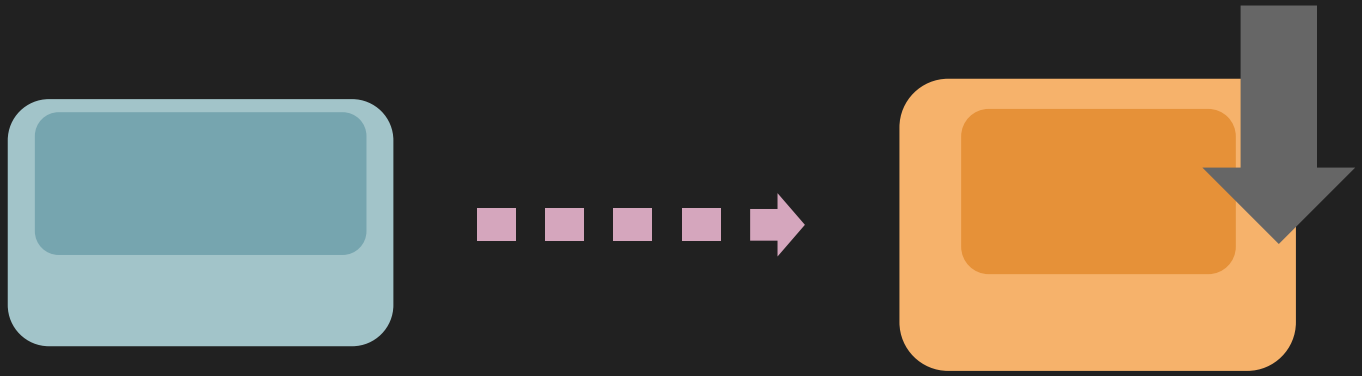
fit vs fit_generator



Simple generator

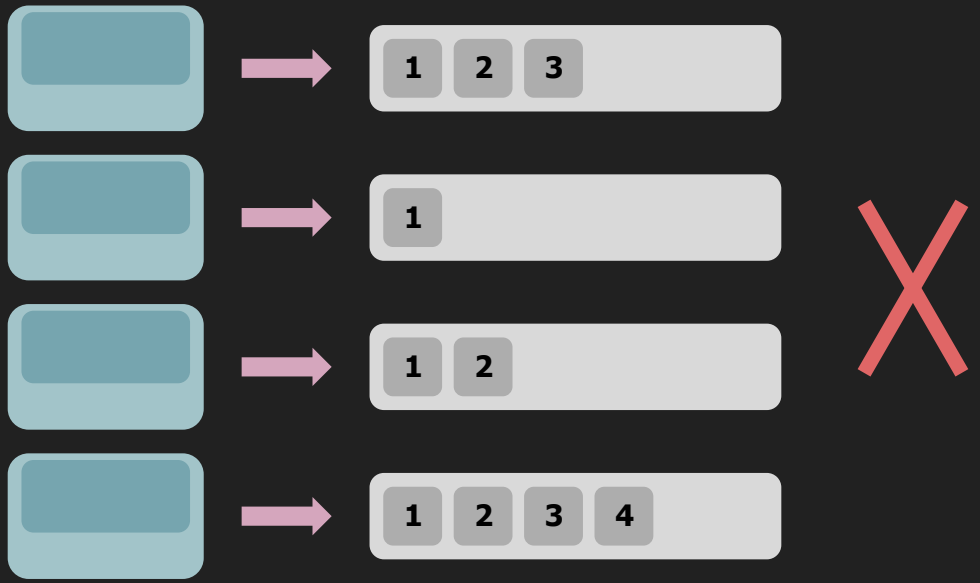
```python
def generator(x_data, y_data, batch_size):
  size = len(x_data)

  while True:
    # shuffle datasets
    idx = np.random.permutation(size)
    x_data = x_data[idx]
    y_data = y_data[idx]

    for i in range(size // batch_size):
      x_batch = x_data[i*batch_size: (i+1)*batch_size]
      y_batch = y_data[i*batch_size: (i+1)*batch_size]

      yield x_batch, y_batch

...

# use generator in fit()
model.fit(generator(x_train, y_train, batch_size), epochs=5,
          steps_per_epoch=len(x_train)//batch_size,
          validation_data=generator(x_test, y_test, batch_size),
          validation_steps=len(x_test)//batch_size)
```

데이터를 그때그때 가져오면
병목이 생길 수 있다



여러 개의 **generator**를 돌려서
병목을 해결할 수 있다

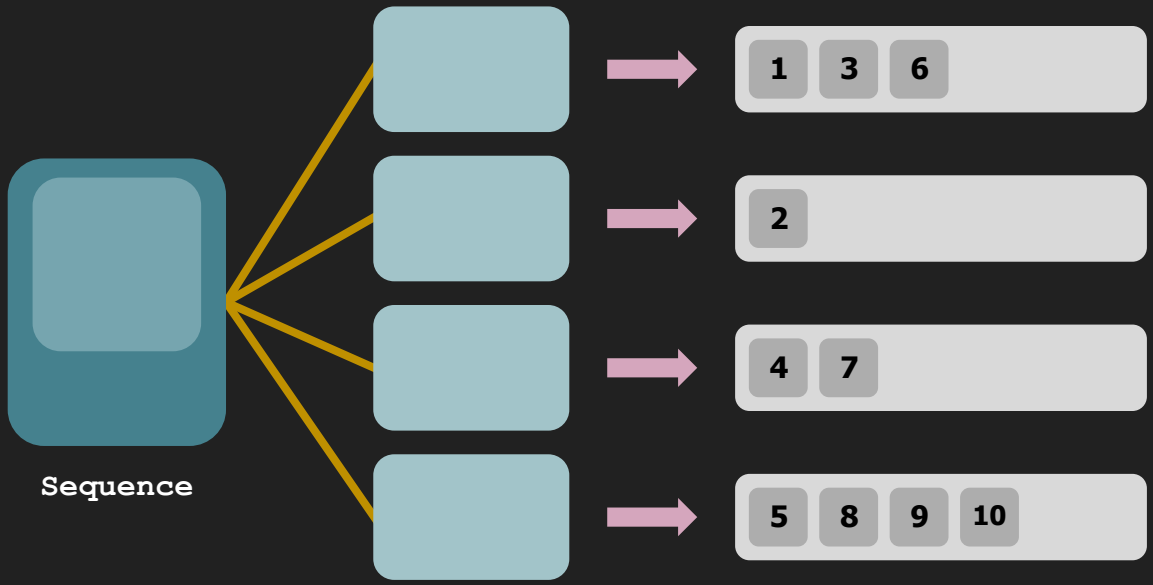
하지만 multi-processing을 하면...



데이터가 중복되는 문제가 생긴다!

thread-unsafe 하다

keras.utils.Sequence



데이터 순서를 지킬 수 있어
thread-safe 하다

Subclassing Sequence

```python
# subclass keras.utils.Sequence
class MNISTSequence(Sequence):
  def __init__(self, x_data, y_data, batch_size):
    self.x_data = x_data
    self.y_data = y_data
    self.batch_size = batch_size
    self.indices = np.random.permutation(len(x_data))

  def __len__(self):
    return len(x_data) // batch_size

  def __getitem__(self, idx):
    batch_idx = self.indices[idx*self.batch_size: (i+1)*self.batch_size]
    x_batch = self.x_data[batch_idx]
    y_batch = self.y_data[batch_idx]

    # you can insert preprocess codes here
    x_batch = (x_batch / 255.).reshape(-1, 28, 28, 1)
    y_batch = to_categorical(y_batch, 10)

    return x_batch, y_batch

  def on_epoch_end(self):
    self.indices = np.random.permutation(len(x_data))
```

Multi-processing example

```
# set datasets
train_generator = MNISTSequence(x_train, y_train, batch_size)
test_generator = MNISTSequence(x_train, y_train, batch_size)

# prepare and train model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_generator, epochs=5, steps_per_epoch=len(train_generator),
          validation_data=test_generator, validation_steps=len(test_generator),
          max_queue_size=50, workers=4, use_multiprocessing=True)
```

Multi-processing이 항상 빠른 것은 아니다
상황에 따라 사용 여부를 결정한다

대용량 데이터셋을 처리하기 어렵다면?

keras.utils.HDF5Matrix

```python
import h5py
from keras.utils import HDF5Matrix

# create hdf5 files and store datasets
with h5py.File('mnist_train.h5', 'w') as f:
  f.create_dataset('image', x_train.shape, dtype='float32')
  f.create_dataset('label', y_train.shape, dtype='float32')
  f['image'][:] = x_train
  f['label'][:] = y_train

...

# prepare dataset
x_train_h5 = HDF5Matrix('mnist_train.h5', 'image')
y_train_h5 = HDF5Matrix('mnist_train.h5', 'label')

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

# train with HDF5: shuffle="batch" is nedded
model.fit(x_train_h5, y_train_h5, batch_size, epochs=10, shuffle='batch')
```

File I/O와 동적 로드를 자동으로 해 준다

직접 구현하지 않고 편하게 쓰고 싶다면?

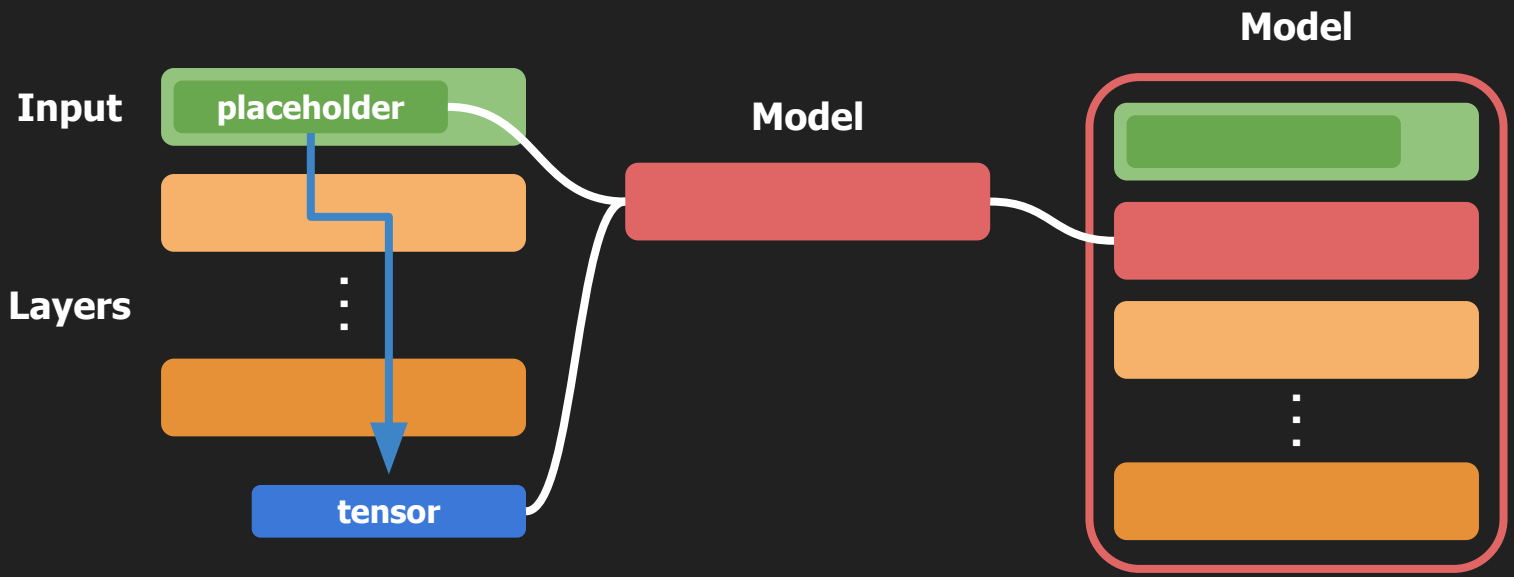
tf.data.Dataset

```
# prepare dataset
train_dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_dataset = train_dataset.shuffle(buffer_size=1024).batch(batch_size)

test_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test))
test_dataset = test_dataset.batch(batch_size)

# prepare and train model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_dataset, batch_size=batch_size, epochs=5, validation_data=test_dataset)
```

shuffle, pipelining, map 등 다양한 기능
tf.keras부터는 그대로 **fit()**에 넣어줄 수 있다

모델을 멋지게 설계하자

Keras Layer & Model 구조



Model과 Layer는 비슷하다!

Keras에는 많은 **Layer**가 있지만
내가 필요한 게 없을 수 있다

어떻게 새로운 **Layer**를 만들 수 있을까?

Custom Layer

```python
from keras import backend as K
from keras.layers import Layer

class MyLayer(Layer):

    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(MyLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        # Create a trainable weight variable for this layer.
        self.kernel = self.add_weight(name='kernel',
                                      shape=(input_shape[1], self.output_dim),
                                      initializer='uniform',
                                      trainable=True)
        super(MyLayer, self).build(input_shape)  # Be sure to call this at the end

    def call(self, x):
        return K.dot(x, self.kernel)

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.output_dim)
```

build(), call(), compute_output_shape()을
구현해 주어야 한다

이 방법이 귀찮다면?

Lambda Layer

```
import keras.backend as K

def interpolate(a, b, alpha=.5):
  return a * alpha + b * (1 - alpha)

...

x1 = ...
x2 = ...
out = Lambda(lambda x: interpolate(x[0], x[1]),
             name='interpolate')([x1, x2])
```

tensor를 받아 **tensor**를 내는 함수를 넣어준다

Lambda Layer를 이용하면
전처리를 GPU에 맡길 수 있다

```
import tensorflow as tf

def preprocess_signal(x, n_fft, max_val=1.):
  normalized = x / max_val
  stft = tf.signal.stft(x, n_fft, n_fft // 4)
  return stft

...

out = Lambda(lambda x: preprocess_signal(x, 2048),
             name='preprocess_signal')(before)
```

Tensorflow의 많은 기능을
Keras에서 쉽게 활용할 수 있다

Custom Model

```python
from keras.layers import *
from keras.models import Model

class SimpleMLP(Model):

    def __init__(self, use_bn=False, use_dp=False, num_classes=10):
        super(SimpleMLP, self).__init__(name='mlp')
        self.use_bn = use_bn
        self.use_dp = use_dp
        self.num_classes = num_classes

        self.dense1 = Dense(32, activation='relu')
        self.dense2 = Dense(num_classes, activation='softmax')
        if self.use_dp:
            self.dp = Dropout(0.5)
        if self.use_bn:
            self.bn = BatchNormalization(axis=-1)

    def call(self, inputs):
        x = self.dense1(inputs)
        if self.use_dp:
            x = self.dp(x)
        if self.use_bn:
            x = self.bn(x)
        return self.dense2(x)

model = SimpleMLP()
model.compile(...)
model.fit(...)
```

Keras Model을 subclassing

call() 함수를 구현한다

Model을 Layer처럼 쓰자

```
# define Encoder
x = Input(shape=input_shape)
...
enc = Model(x, enc_out, name='encoder')

# define Decoder
z = Input(shape=enc.output_shape)
...
dec = Model(z, dec_out, name='decoder')

# connect two models and define AutoEncoder
autoencoder = Model(x, dec(enc(x)), name='autoencoder')
```

sub-model을 구현한 뒤에 연결해서
큰 구조의 **Model**을 설계할 수 있다

Loss나 Metric도
원하는대로 만들 수 있다

```python
import keras.backend as K

def mean_pred(y_true, y_pred):
    return K.mean(y_pred)

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy', mean_pred])
```

(y_true, y_pred) 를 인자로 받으면 된다

무조건 (y_true, y_pred) 이어야 한다

또 다른 인자를 넘겨주고 싶은 경우엔?

```python
import keras.backend as K
import math

def psnr(max_value):
  a = 1. / math.log(10.0)

  def get_psnr_val(y_true, y_pred):
    mse = K.mean(K.square(y_true - y_pred), axis=-1)
    return 20.*a * K.log(max_value) - 10.*a * K.log(mse)

  return get_psnr_val
```

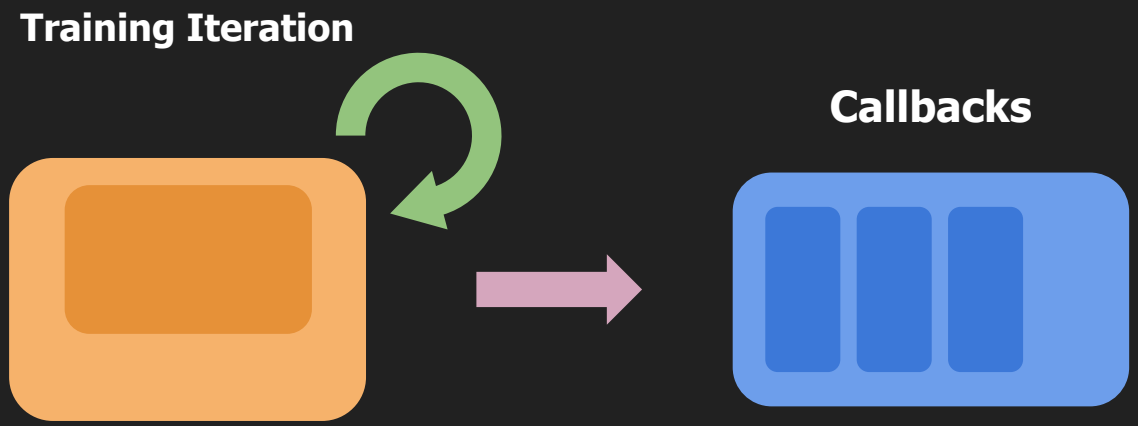
Nested function을 이용한다

모델을 멋지게 학습하자

학습이 잘 되는 건가..?

학습 중에도 모델의 결과를 눈으로 보고 싶다!

학습 중간중간 실행되는
Callback이 있다



모델 저장, **Tensorboard**로 보기, 학습률 조정 등
학습 시에 도움이 되는 기능이 많다

Callback example

fit()에 callbacks 옵션을 주면 된다

```
from keras.callbacks import ModelCheckpoint, TensorBoard

...

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

# train with callbacks
callbacks = [
  ModelCheckpoint('model.h5', monitor='val_loss', period=5),
  TensorBoard(log_dir='./logs', histogram_freq=1),
]
model.fit(x_train, y_train, batch_size, epochs=10,
          validation_data=(x_test, y_test),
          callbacks=callbacks)
```

만약 Callback 중 필요한 기능이 없으면?

Custom Callback

```python
from keras.callbacks import Callback, LambdaCallback

# subclassing custom callback
class MyCallback(Callback):
  def on_batch_begin(epoch, logs):
    ...
  def on_batch_end(epoch, logs):
    ...
  def on_epoch_begin(epoch, logs):
    ...
  def on_epoch_end(epoch, logs):
    ...
  def on_train_begin(logs):
    ...
  def on_train_end(logs):
    ...

my_callback = MyCallback()

# or you can just give function in LambdaCallback
my_callback = LambdaCallback(on_epoch_end=
                             lambda epoch, logs: eval_model(epochs))
```

6개의 callback 함수들을
상황에 맞게 구현해주면 된다

Callback으로는 부족하다!

학습 과정을 더 자세히 보고싶다면?

```
...

for epoch in range(num_epoch):

  # on_epoch_begin logics

  for step, (x_batch, y_batch) in enumerate(x_train, y_train):

    generated = generator.predict(x_train)

    d_loss_fake = d_trainer.train_on_batch(generated, fake_label)
    d_loss_real = d_trainer.train_on_batch(y_batch, valid_label)
    d_loss = .5 * d_loss_fake + .5 * d_loss_real

    g_loss = g_trainer.train_on_batch(generated, valid_label)

    print('d_loss: {} - g_loss: {}'.format(d_loss, g_loss))

  # on_epoch_end logics
```

fit()을 사용하지 않고
직접 학습 과정을 짤 수 있다

tf.GradientTape를 쓰자

Pytorch와 비슷한 방법으로 사용이 가능하다

```
optimizer = keras.optimizers.Adam(lr=lr)
loss = keras.loss.SparseCategoricalCrossentropy()

for epoch in range(num_epoch):

  # on_epoch_begin logics

  for step, (x_batch, y_batch) in enumerate(train_dataset):

    with tf.GradientTape() as tape:
      logits = model(x_batch)
      loss_val = loss(y_batch, logits)

    grads = tape.gradient(loss_val, model.trainable_weights)
    optimizer.apply_gradients(grads, model.trainable_weights)

  # on_epoch_end logics
```

좀 더 편리하고 자유로운 방식으로
학습 로직 구현이 가능하다

Multi-GPU에서 학습은 어떻게 할까?

```python
from keras.utils import multi_gpu_model

...

single_model = model
multi_model = multi_gpu_model(model, gpus=4)

multi_model.compile(optimizer='adam', loss='mse')
multi_model.fit(x_train, y_train, batch_size, epochs=10,
                validation_data=(x_test, y_test))

# store with single model
single_model.save('model.h5')
```

간단한 함수 하나로 해결이 가능하다
multi_gpu_model()을 쓰면 된다

결과를 멋지게 활용하자

ONNX

Open Neural Network eXchange format

여러 딥러닝 프레임워크 간에
모델을 전환할 수 있다

Keras to ONNX

onnxmltools 라이브러리를 사용하면 된다

```
import onnxmltools
from keras.models import load_model

input_keras_model = 'model.h5'
output_onnx_model = 'model.onnx'

keras_model = load_model(input_keras_model)
onnx_model = onnxmltools.convert_keras(model)
onnxmltools.utils.save_model(onnx_model, output_onnx_model)
```

현재 **ONNX to Keras**는 불가능하고

Tensorflow 모델로 바꿀 수 있다

Convert for iOS

coremltools 라는 라이브러리를 쓴다


```
import coremltools
from keras.models import load_model

output_labels = [...]
your_model = coremltools.converters.keras.convert('your_model.h5',
                                                   input_names=['image'],
                                                   output_names=['output'],
                                                   class_labels=output_labels,
                                                   image_input_names='image')

your_model.save('model.mlmodel')
```

Convert for Android

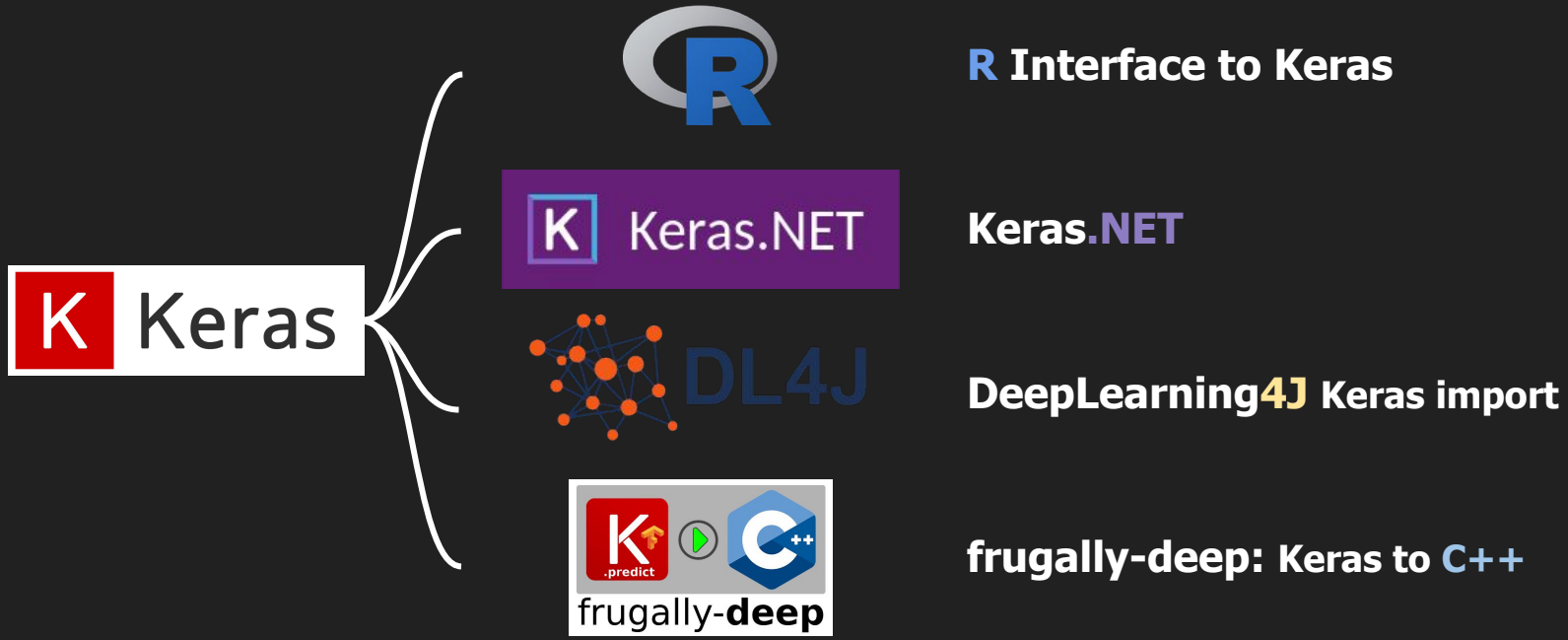
Tensorflow를 거쳐 변환한다

```
import tensorflow as tf

...

model.compile(optimizer='sgd', loss='mean_squared_error')
model.fit(x, y, epochs=50)

# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model.
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
```

TFLite를 사용하면 더 편리하다
Android, iOS 모두 변환이 된다!

Keras 모델을 다른 언어에서 쓰려면?

이를 지원하는 여러 도구가 있다!



R Interface to Keras

Keras.NET

DeepLearning4J Keras import

frugally-deep: Keras to C++

Keras를 쓸 때 유용한 프로젝트
시각화, 튜닝 자동화 등 다양하게 있다

keras-vis
모델 디버깅을 도와주는 다양한 시각화 테크닉이 구현되어 있다

keras-applications
다양한 모델들의 구조와 **pretrained weights**을 제공한다

AutoKeras
AutoML을 **Keras**에서 할 수 있는 라이브러리

Keras Tuner
Hyper-parameter tuning을 쉽게 할 수 있도록 도와준다

References

tf.keras 훑어보기	https://www.tensorflow.org/guide/keras/overview?hl=ko
keras Sequence	https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly
HDF5Matrix	https://nuxlear.tistory.com/4
tf.data.Dataset	https://www.tensorflow.org/guide/keras/train_and_evaluate#training_evaluation_from_tfdata_datasets
Custom Layer	https://keras.io/layers/writing-your-own-keras-layers/
Model Subclassing	https://keras.io/models/about-keras-models/
Callbacks	https://keras.io/callbacks/
Custom Training (GAN)	https://github.com/eriklindernoren/Keras-GAN/blob/master/dcgan/dcgan.py
tf.GradientTape	https://www.tensorflow.org/guide/keras/train_and_evaluate#using_the_gradienttape_a_first_end-to-end_example
keras-onnx	https://github.com/onnx/keras-onnx
Core ML tools	https://heartbeat.fritz.ai/using-coremltools-to-convert-a-keras-model-to-core-ml-for-ios-d4a0894d4aba
TFLite	https://www.tensorflow.org/lite/convert
TFLite for Android	https://medium.com/@vvalouch/from-keras-to-android-with-tensorflow-lite-7581368aa23e
TF 2.0 Quick Tutorial	https://github.com/tensorflow/docs/blob/master/site/en/tutorials/quickstart/advanced.ipynb

QnA

Thank You