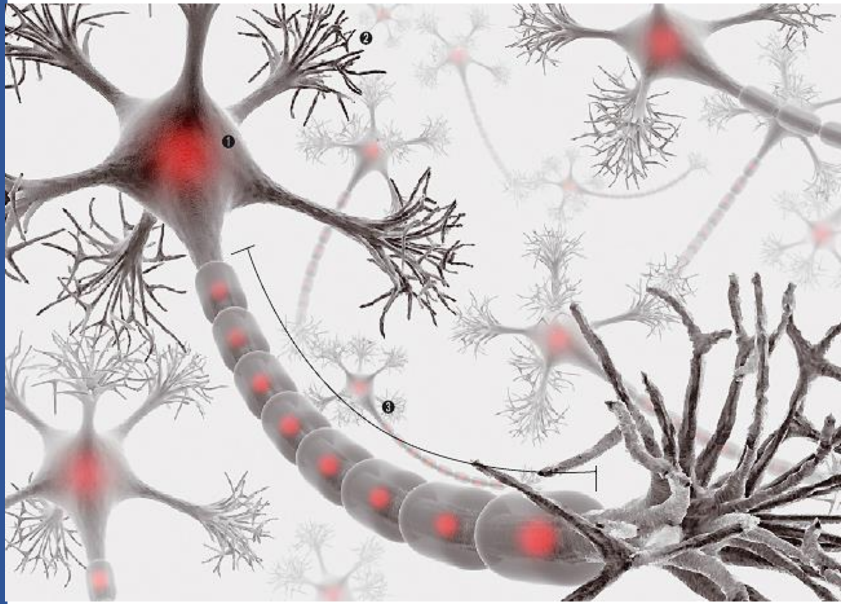
Keras

학습 목표

- Tensorflow의 High-Level Wrapper인 Keras의 사용법을 이해한다.
- .

주요 내용

1. 클래스 구조
2. 모델 정의 방법
3. 모델 훈련 방법
4. 활용 클래스

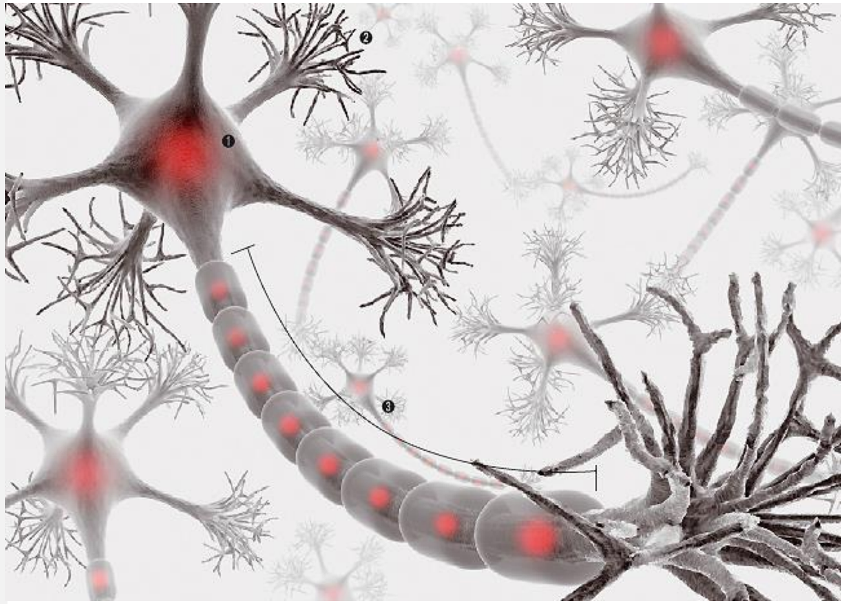


Keras

- **High-Level 신경망 스펙** (https://keras.io) (2015. 03)
- TensorFlow 1.2 : tf.contrib.keras로 추가됨
- TensorFlow 1.4 : tf.keras로 한단계 승격 (tf.layers → tf.keras)
- TensorFlow 2.0 : 1st Class Python API
- tf.layer, tf.contrib.layers(Slim)는 Deprecated 됨
- Keras 2.3.x 가 multi-backend Keras의 마지막 major release
  따라서, tf.keras를 사용하는 것이 좋음

Keras in TensorFlow2.0 by 박해선님

1 클래스 계층 구조



Class Hierarchy

변수 컨테이너 (tf.Variable)

variables(), trainable_variables()

계층 정의 (파라미터, Forward Pass)

__call__() → build() → add_weights()
|→ call()
add_loss()

신경망 계층 통합

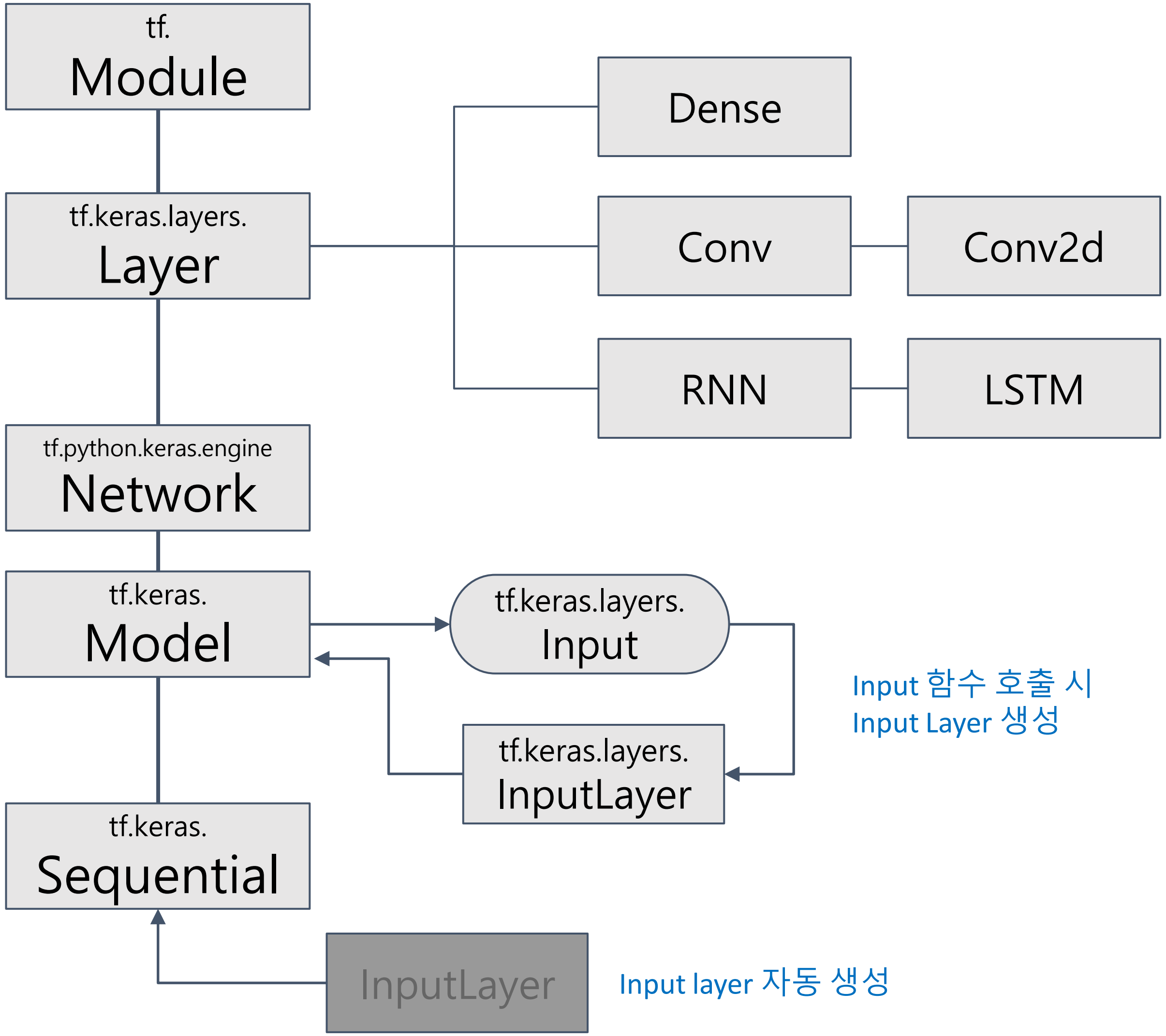
layers(), summary(), save()

모델 훈련/검증/테스트

compile(), fit(), evaluate(), predict()

순차 모델 구성

add()

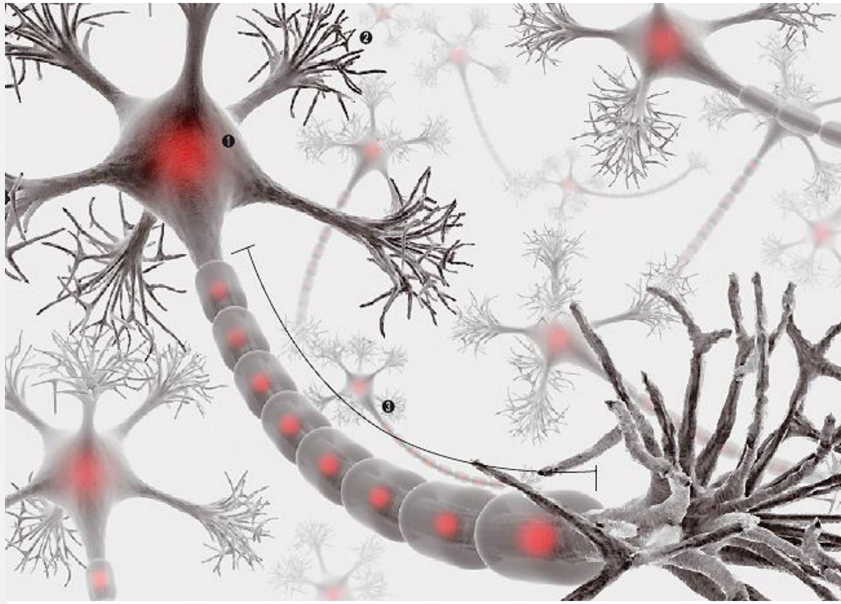


Input 함수 호출 시
Input Layer 생성

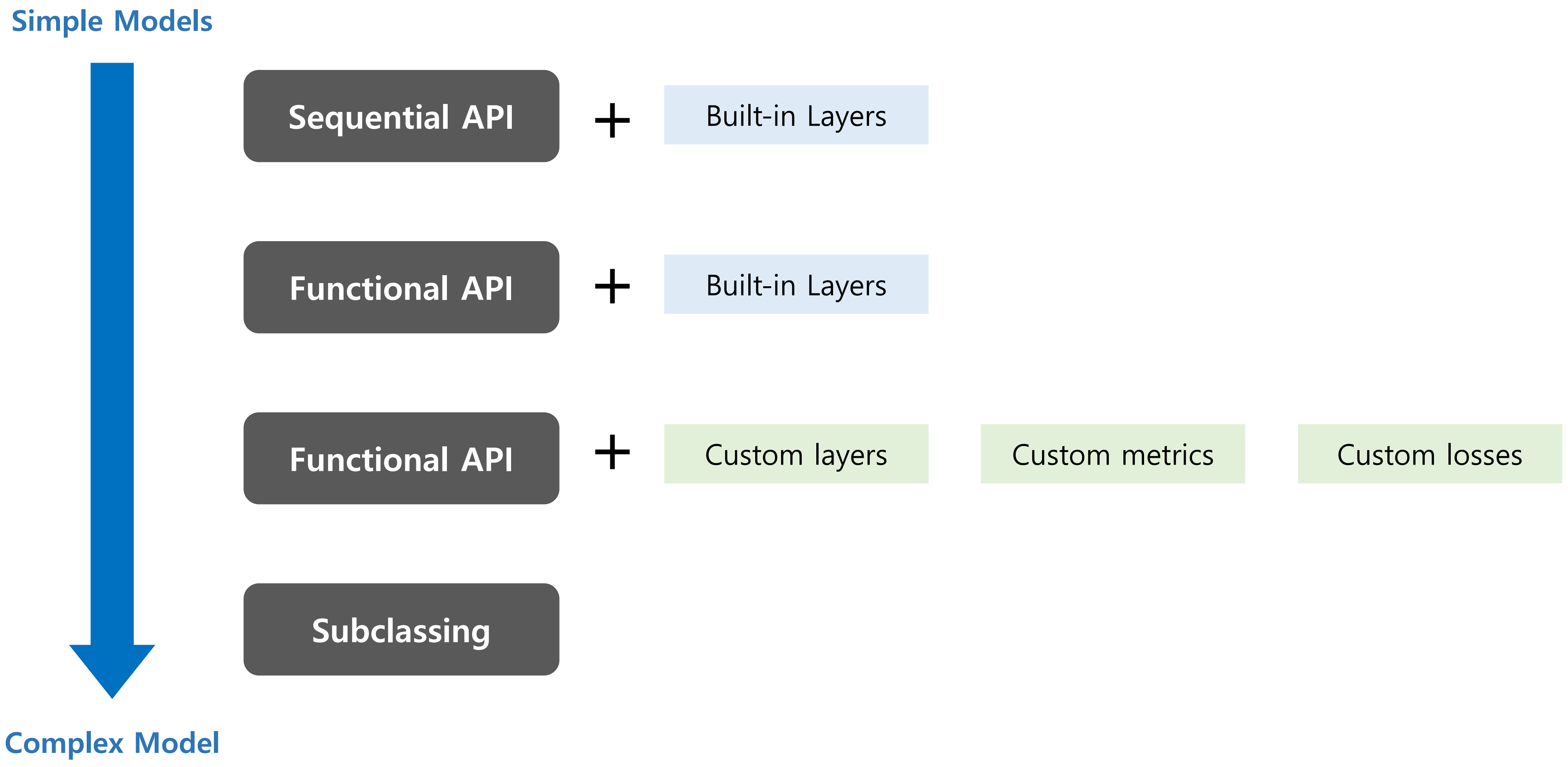
Input layer 자동 생성

Keras in TensorFlow2.0 by 박해선님

2 모델 정의 방법



Keras 모델 정의



Simple Models

Sequential API + Built-in Layers

Functional API + Built-in Layers

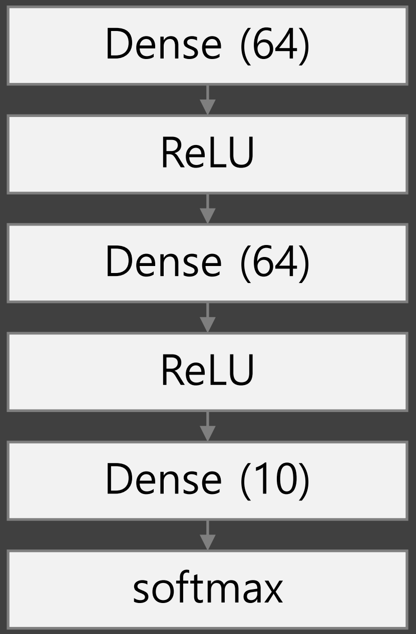
Functional API + Custom layers Custom metrics Custom losses

Subclassing

Complex Model

Sequential API

```
from tensorflow import tf

model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

# 훈련 설정
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
                loss='categorical_crossentropy',
                metrics=['accuracy'])

# 모델 훈련
model.fit(train_data, labels, epochs=10, batch_size=32)

# 모델 평가
model.evaluate(test_data, labels)

# 샘플 예측
model.predict(new_sample)
```



```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64),
    tf.keras.layers.Dense(64),
    tf.keras.layers.Dense(10),
])
```

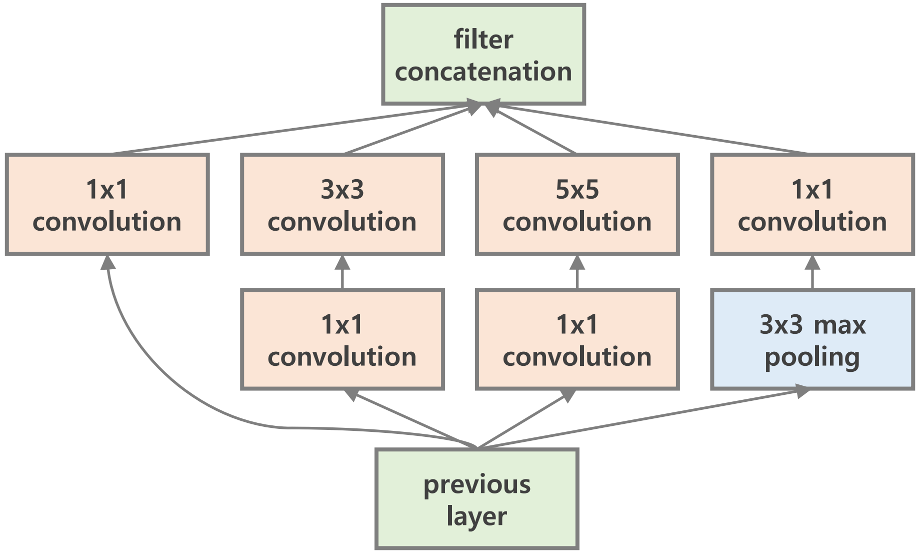
Functional API

```
from tensorflow import tf

# 입력과 출력을 연결해서 임의의 모델 그래프 생성
input = tf.keras.Input(shape=(784,), name='img’)  # 입력 플레이스 홀더 반환
h1 = tf.keras.layers.Dense(64, activation='relu')(inputs) # 각 계층 별로 Tensor를 전달하고 리턴 받음
h2 = tf.keras.layers.Dense(64, activation='relu')(h1)
output = tf.keras.layers.Dense(10, activation='softmax')(h2)

# 모델 생성
model = tf.keras.Model(input, output) # 입력 Tensor와 Output Tensor를 모델에 지정

# 훈련 설정
model.compile(optimizer=tf.keras.optimizers.Adam(0.001),
                      loss='categorical_crossentropy',
                      metrics=['accuracy’])
…
```

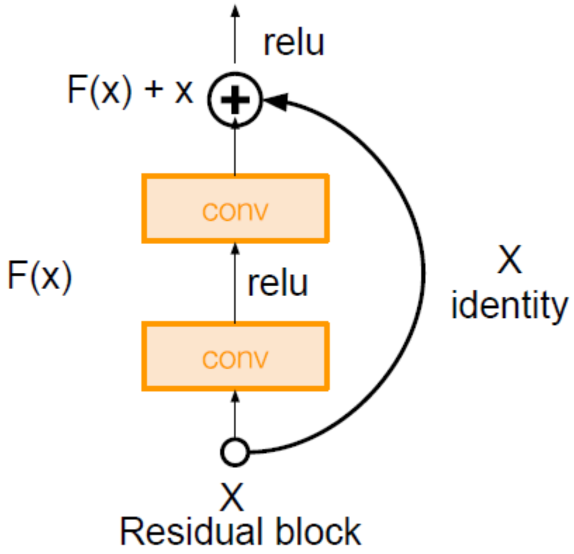
Keras in TensorFlow2.0 by 박해선님

Functional API

“Inception module”



“Residual block”



- 다중 입력 모델
- 다중 출력 모델
- 층을 공유하는 모델 (동일한 층을 여러 번 호출합니다)
- 데이터 흐름이 차례대로 진행되지 않는 모델 (예를 들면 잔차 연결(residual connections)).

Custom Layer

```python
from tensorflow import tf
class MyLayer(tf.keras.layers.Layer):

    def __init__(self, units, activation=None, **kwargs):
        self.units = units
        self.activation = keras.activations.get(activation)
        super().__init__(**kwargs)

    def build(self, input_shape):
        self.weight = self.add_weight(name='kernel',
                                      shape=(input_shape[1], self.units),
                                      initializer='uniform')
        self.bias = self.add_weight(name='bias',
                                    shape=(self.units,),
                                    initializer='zeros')
        super().build(input_shape)

    def call(self, X):
        z = tf.matmul(X, self.weight) + self.bias
        return self.activation(z)
```

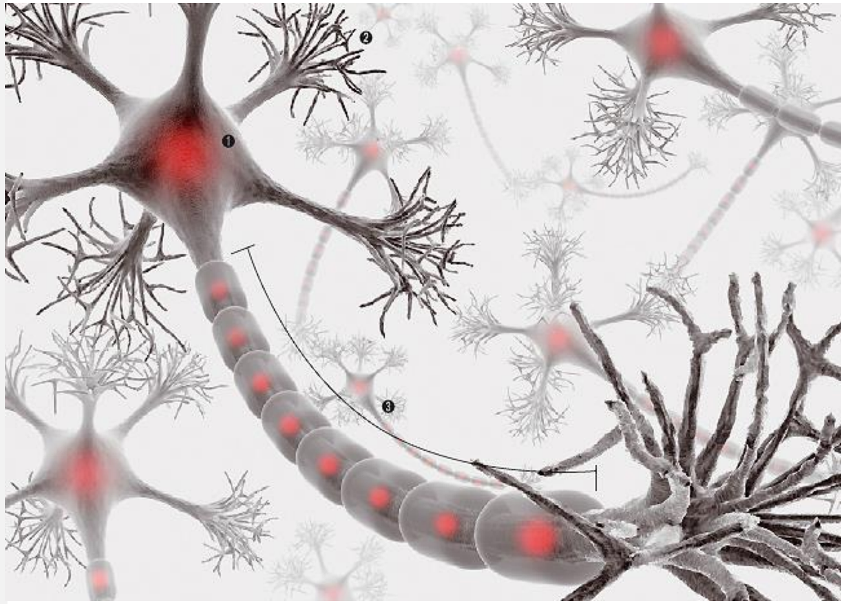
Keras in TensorFlow2.0 by 박해선님

Custom Model

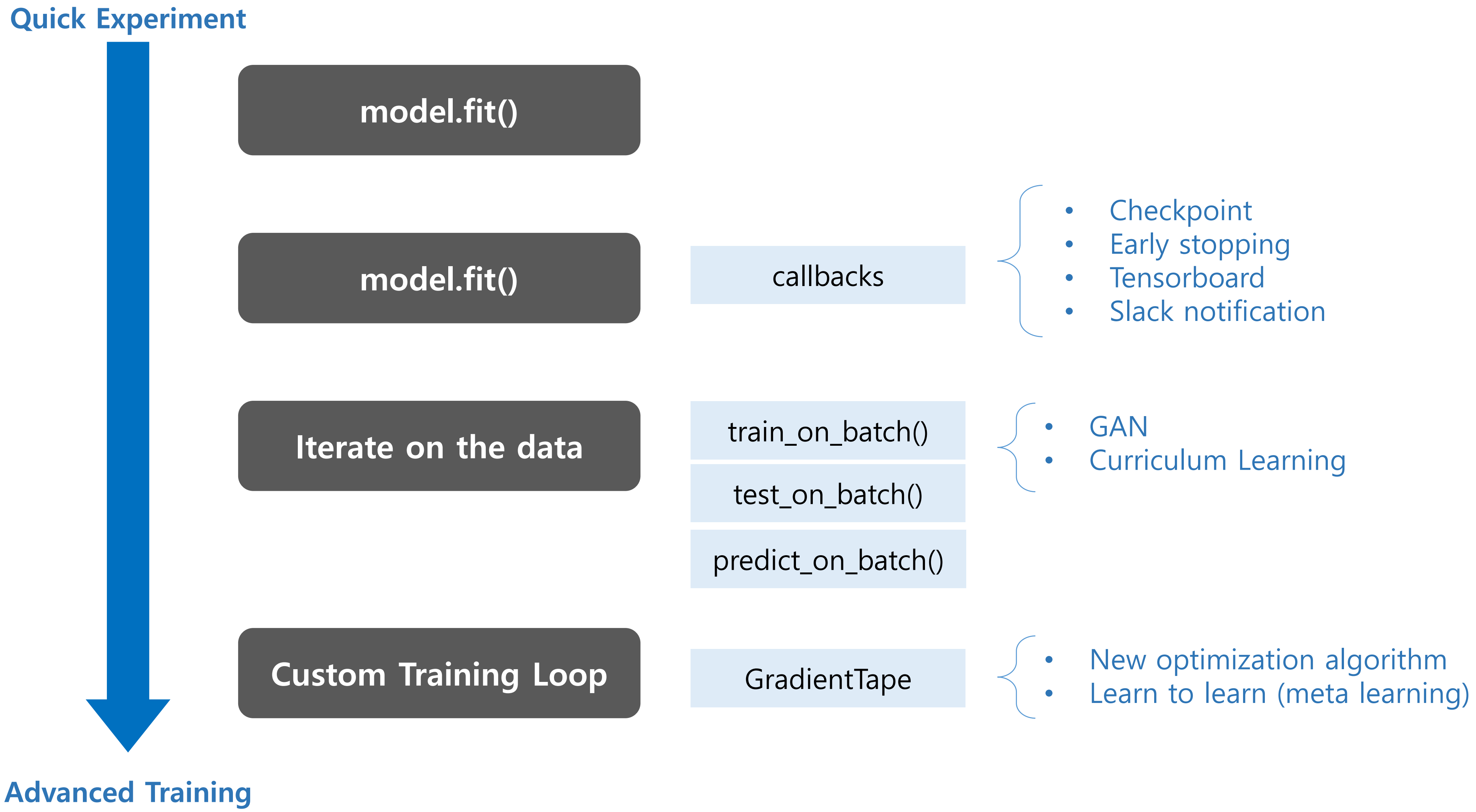
```
from tensorflow import tf

class MyModel(tf.keras.Model):

    def __init__(self, **kwargs):
        self.hidden = MyLayer(10, activation="relu")
        self.output = MyLayer(1)
        super().__init__(**kwargs)

    def call(self, input):
        h = self.hidden(input)
        return self.output(h)

model = MyModel()
```

Keras in TensorFlow2.0 by 박해선님

3 모델 훈련 방법



Training 방식



Quick Experiment

model.fit()

model.fit()

callbacks

- Checkpoint
- Early stopping
- Tensorboard
- Slack notification

Iterate on the data

train_on_batch()

test_on_batch()

predict_on_batch()

- GAN
- Curriculum Learning

Custom Training Loop

GradientTape

- New optimization algorithm
- Learn to learn (meta learning)

Advanced Training

model.compile

훈련에 필요한 Optimizer, Loss, Metric을 설정하는 단계

1. 이름으로 지정 (Default 값으로 실행할 때)

회귀 모델 예시

```
model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
              loss='mse',      # 평균 제곱 오차
              metrics=['mae'])  # 평균 절댓값 오차
```

2. 객체를 생성해서 전달 (파라미터를 지정할 필요가 있을 때)

분류 모델 예시

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(0.01),
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=[tf.keras.metrics.CategoricalAccuracy()])
```

model.fit

모델을 고정된 epoch 수로 훈련

```
history = model.fit( train_data, train_labels,
                     epochs=1000, validation_split = 0.2, verbose=0,
                     callbacks=[Earlystopping(),
                                Tensorboard(),
                                ModelCheckpoint()])
```

- **batch_size**: 배치 크기 (default 32)
- **epochs**: 총 epoch 수 (epoch는 training set을 한번 실행하는 단위)
- **validation_split**: training set에서 validation set으로 사용할 비율 ((0,1) 사이의 값)
- **verbose**: 훈련 진행 상황 모드 0 = silent, 1 = progress bar, 2 = one line per epoch
- **callbacks**: 훈련하면서 실행할 콜백 리스트

tf.GradientTape

```
@tf.function
def train_step(input, target):
 with tf.GradientTape() as tape:
   # forward Pass
   predictions = model(input)
   # compute the loss
   loss = tf.reduce_mean(
            tf.keras.losses.sparse_categorical_crossentropy(
            target, predictions, from_logits=True))
 # compute gradients
 grads = tape.gradient(loss, model.trainable_variables)
 # perform a gradient descent step
 optimizer.apply_gradients(zip(grads, model.trainable_variables))
 return loss
```

Forward Pass

Gradient 계산

Parameter Update

Keras + eager mode

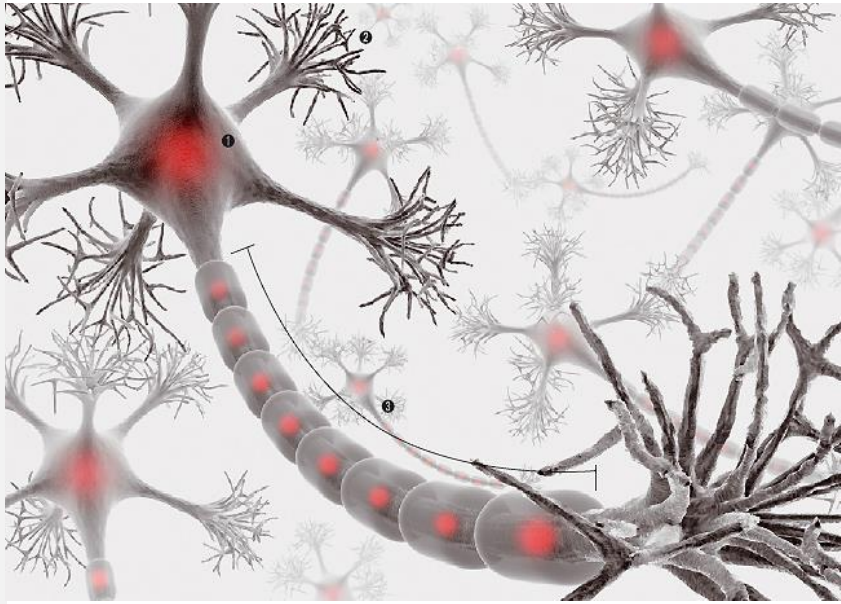
- Keras는 default로 @tf.function 사용

```
class CustomModel(tf.keras.models.Model):

  @tf.function
  def call(self, input_data):
    if tf.reduce_mean(input_data) > 0:
      return input_data
    else:
      return input_data // 2
```

- Eager mode를 사용하려면 다음 설정을 해야 함
  - model = Model(dynamic=True) or
  - model.compile(..., run_eagerly=True)

Keras in TensorFlow2.0 by 박해선님

4 활용 클래스



참고 tf.keras.layers

- **class Conv2D:** 2D convolution layer (e.g. spatial convolution over images).
- **class Dense**: Just your regular densely-connected NN layer.

Layer

- **class Flatten**: Flattens the input. Does not affect the batch size.
- class Reshape: Reshapes an output to a certain shape.
- class InputLayer: Layer to be used as an entry point into a Network (a graph of layers).

모양 변경

- class **MaxPool2D**: Max pooling operation for spatial data.
- class AveragePooling2D: Average pooling operation for spatial data.
- **class GlobalAveragePooling2D**: Global average pooling operation for spatial data.

Pooling

- **class BatchNormalization:** Normalize and scale inputs or activations. (Ioffe and Szegedy, 2014).
- class **Dropout**: Applies Dropout to the input.

정규화

- **class Embedding**: Turns positive integers (indexes) into dense vectors of fixed size.
- class SimpleRNN: Fully-connected RNN where the output is to be fed back to input.
- **class LSTM**: Long Short-Term Memory layer - Hochreiter 1997.
- class GRU: Gated Recurrent Unit - Cho et al. 2014.

RNN 계열

https://www.tensorflow.org/api_docs/python/tf/keras/layers

참고 tf.keras.layers

- class Softmax: Softmax activation function.
- **class ReLU**: Rectified Linear Unit activation function.
- class LeakyReLU: Leaky version of a Rectified Linear Unit.
- class ELU: Exponential Linear Unit.

Activation Function

https://www.tensorflow.org/api_docs/python/tf/keras/layers

참고 tf.keras.optimizers

- class SGD: Stochastic gradient descent and momentum optimizer.
- class Adagrad: Optimizer that implements the Adagrad algorithm.
- **class RMSprop**: Optimizer that implements the RMSprop algorithm.
- **class Adam**: Optimizer that implements the Adam algorithm.

https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

참고 tf.keras.losses

- **class MeanSquaredError**: Computes the mean of squares of errors between labels and predictions.
- class MeanAbsoluteError: Computes the mean of absolute difference between labels and predictions.

- **class BinaryCrossentropy:** Computes the cross-entropy loss between true labels and predicted labels.
- **class CategoricalCrossentropy**: Computes the crossentropy loss between the labels and predictions.
- **class SparseCategoricalCrossentropy**: Computes the crossentropy loss between the labels and predictions.

https://www.tensorflow.org/api_docs/python/tf/keras/losses

참고 tf.keras.metrics

- **class Accuracy**: Calculates how often predictions matches labels.
- **class MeanAbsoluteError**: Computes the mean absolute error between the labels and predictions.
- **class MeanSquaredError**: Computes the mean squared error between y_true and y_pred.

https://www.tensorflow.org/api_docs/python/tf/keras/metrics

참고 tf.keras.callbacks

- **class EarlyStopping**: Stop training when a monitored quantity has stopped improving.
- **class ModelCheckpoint**: Save the model after every epoch.
- class TensorBoard: Enable visualizations for TensorBoard.
- class LearningRateScheduler: Learning rate scheduler.

https://www.tensorflow.org/api_docs/python/tf/keras/callbacks

Thank you!