케라스 신경망 모델 활용법

주요 내용

- 다양한 신경망 모델 구성법
- 신경망 모델과 층의 재활용
- 신경망 모델 훈련 옵션: 콜백과 텍서보드

신경망 모델 구성법 1: `Sequential` 모델 활용

- `Sequential` 모델은 층으로 스택을 쌓아 만든 모델이며 가장 단순함
- 한 종류의 입력값과 한 종류의 출력값만 사용 가능
- 순전파: 지정된 층의 순서대로 적용

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    layers.Dense(64, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

summary() 메서드

```
>>> model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	?	0 (unbuilt)
dense_1 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

Input() 함수

```
model = keras.Sequential([
    keras.Input(shape=(784,)),
    layers.Dense(64, activation="relu"),
    layers.Dense(10, activation="softmax")
])
```

```
>>> model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 64)	50,240
dense_5 (Dense)	(None, 10)	650

Total params: 50,890 (198.79 KB)

Trainable params: 50,890 (198.79 KB)

Non-trainable params: 0 (0.00 B)

층별 가중치 텐서

- 1층의 가중치 행렬(2차원 텐서) 모양: (784, 64)
    - 입력값 특성: 784개
    - 출력값 특성: 64개

```
>>> model.weights[0].shape # 가중치 행렬
TensorShape([784, 64])
```

- 1층의 편향 벡터(1차원 텐서) 모양: (64,)
    - 출력값 특성: 64개

```
>>> model.weights[1].shape # 편향 벡터
TensorShape([64])
```

- 2층의 가중치 행렬(2차원 텐서): (64, 10)
    - 입력값 특성: 64개
    - 출력값 특성: 10개

```
>>> model.weights[2].shape # 가중치 행렬
TensorShape([64, 10])
```

- 2층의 편향 벡터(1차원 텐서) 모양: (10,)
    - 출력값 특성: 10개

```
>>> model.weights[3].shape # 편향 벡터
TensorShape([10])
```

신경망 모델 구성법 2: 함수형 API

```
inputs = keras.Input(shape=(3,), name="my_input")          # 입력층
features = layers.Dense(64, activation="relu")(inputs)     # 은닉층
outputs = layers.Dense(10, activation="softmax")(features) # 출력층
model = keras.Model(inputs=inputs, outputs=outputs)        # 모델 지정
```

```
>>> model.summary()
```

Model: "functional_6"

Layer (type)	Output Shape	Param #
my_input (InputLayer)	(None, 784)	0
dense_10 (Dense)	(None, 64)	50,240
dense_11 (Dense)	(None, 10)	650

Total params: 50,890 (198.79 KB)

Trainable params: 50,890 (198.79 KB)

Non-trainable params: 0 (0.00 B)

다중 입력, 다중 출력 모델

다중 입력과 다중 출력을 지원하는 모델을 구성하는 방법을 예제를 이용하여 설명한다.

- 입력층: 세 개
- 은닉층: 두 개
- 출력층: 두 개

```
vocabulary_size = 10000    # 사용빈도 1만등 인내 단어 사용
num_tags = 100             # 태그 수
num_departments = 4        # 부서 수

# 입력층: 세 개
title = keras.Input(shape=(vocabulary_size,), name="title")
text_body = keras.Input(shape=(vocabulary_size,), name="text_body")
tags = keras.Input(shape=(num_tags,), name="tags")

# 은닉층
features = layers.Concatenate()([title, text_body, tags]) # shape=(None,
10000+10000+100)
features = layers.Dense(64, activation="relu")(features)

# 출력층: 두 개
priority = layers.Dense(1, activation="sigmoid", name="priority")(features)
department = layers.Dense(
    num_departments, activation="softmax", name="department")(features)

# 모델 빌드: 입력값으로 구성된 입력값 리스트와 출력값으로 구성된 출력값 리스트
사용
model = keras.Model(inputs=[title, text_body, tags], outputs=[priority,
department])
```

모델 컴파일

```
model.compile(optimizer="adam",
              loss=["mean_squared_error", "categorical_crossentropy"],
              metrics=[["mean_absolute_error", "mean_squared_error"],
                       ["accuracy", "AUC", "Precision"]])
```

모델 훈련

```
model.fit([title_data, text_body_data, tags_data],
          [priority_data, department_data],
          epochs=10)
```

모델 평가

```
model.evaluate([title_data, text_body_data, tags_data],
               [priority_data, department_data])
```

모델 활용

```
priority_preds, department_preds = model.predict([title_data,
                                                  text_body_data,
                                                  tags_data])
```

- 우선 순위 예측값: 0과 1사이의 확률값

```
>>> priority_preds
array([[1.],
   [1.],
   [1.],
   ...,
   [1.],
   [1.],
   [1.]], dtype=float32)
```

- 처리 부서 예측값: 각 부서별 적정도를 가리키는 확률값

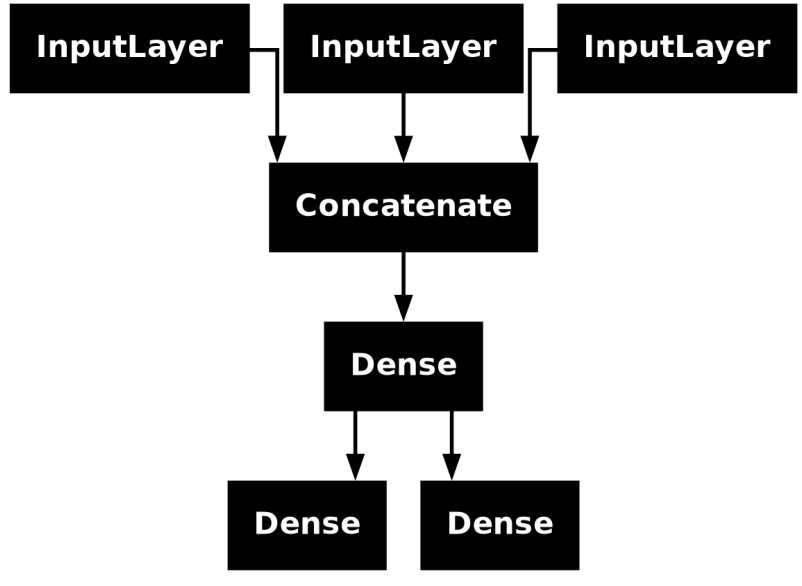
```
>>> department_preds
array([[1.267548e-05, 3.678832e-11, 2.387379e-03, 9.975999e-01],
       [5.863077e-03, 4.932786e-09, 6.003909e-01, 3.937459e-01],
       [1.562561e-03, 3.384366e-07, 2.208202e-02, 9.763551e-01],
       ...,
       [2.978364e-03, 6.375713e-07, 4.778040e-03, 9.922429e-01],
       [2.411681e-05, 3.638920e-10, 3.098509e-01, 6.901249e-01],
       [9.115771e-05, 7.135761e-10, 7.342336e-02, 9.264854e-01]],
      dtype=float32)
```

각각의 요구사항을 처리해야 하는 부서는 `argmax()` 메서드로 확인된다.

```
>>> department_preds.argmax()
array([3, 2, 3, ..., 3, 3, 3])
```

신경망 모델 구조 그래프

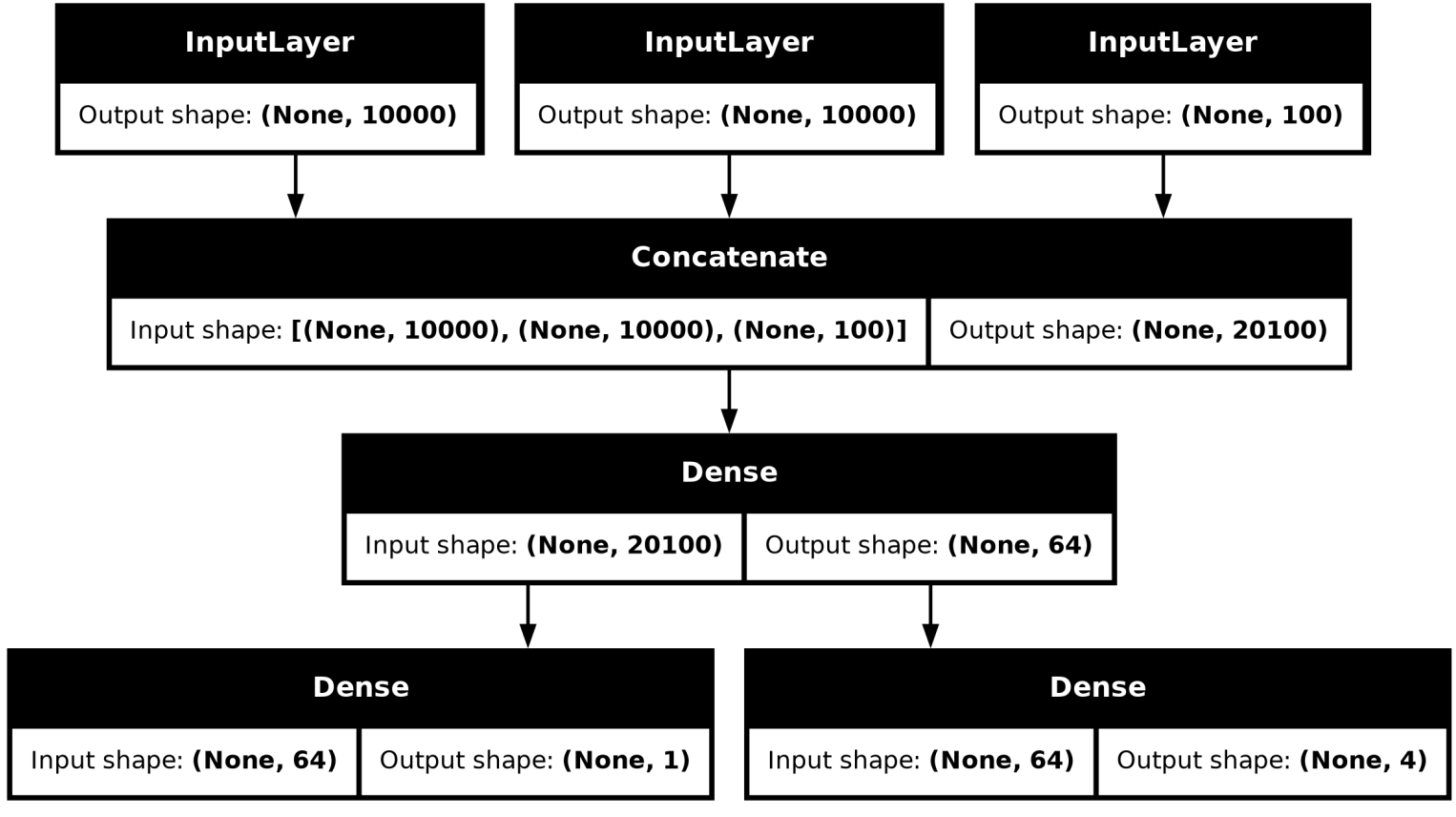
```
>>> keras.utils.plot_model(model, "ticket_classifier.png")
```



`plot_model()` 함수 사용 준비 사항

- `pydot` 모듈 설치: `pip install pydot`
- graphviz 프로그램 설치: https://graphviz.gitlab.io/download/
- 구글 코랩에서는 기본으로 지원됨.

```
>>> keras.utils.plot_model(model, "ticket_classifier_with_shape_info.png", show_shapes=True)
```



신경망 모델 재활용

```
>>> model.layers
[<keras.src.engine.input_layer.InputLayer at 0x7fc3a1313fd0>,
 <keras.src.engine.input_layer.InputLayer at 0x7fc3a13ce450>,
 <keras.src.engine.input_layer.InputLayer at 0x7fc3a13c5990>,
 <keras.src.layers.merging.concatenate.Concatenate at 0x7fc3a13e0d50>,
 <keras.src.layers.core.dense.Dense at 0x7fc3a13a6310>,
 <keras.src.layers.core.dense.Dense at 0x7fc3a12f6850>,
 <keras.src.layers.core.dense.Dense at 0x7fc3a13e2f90>]
```

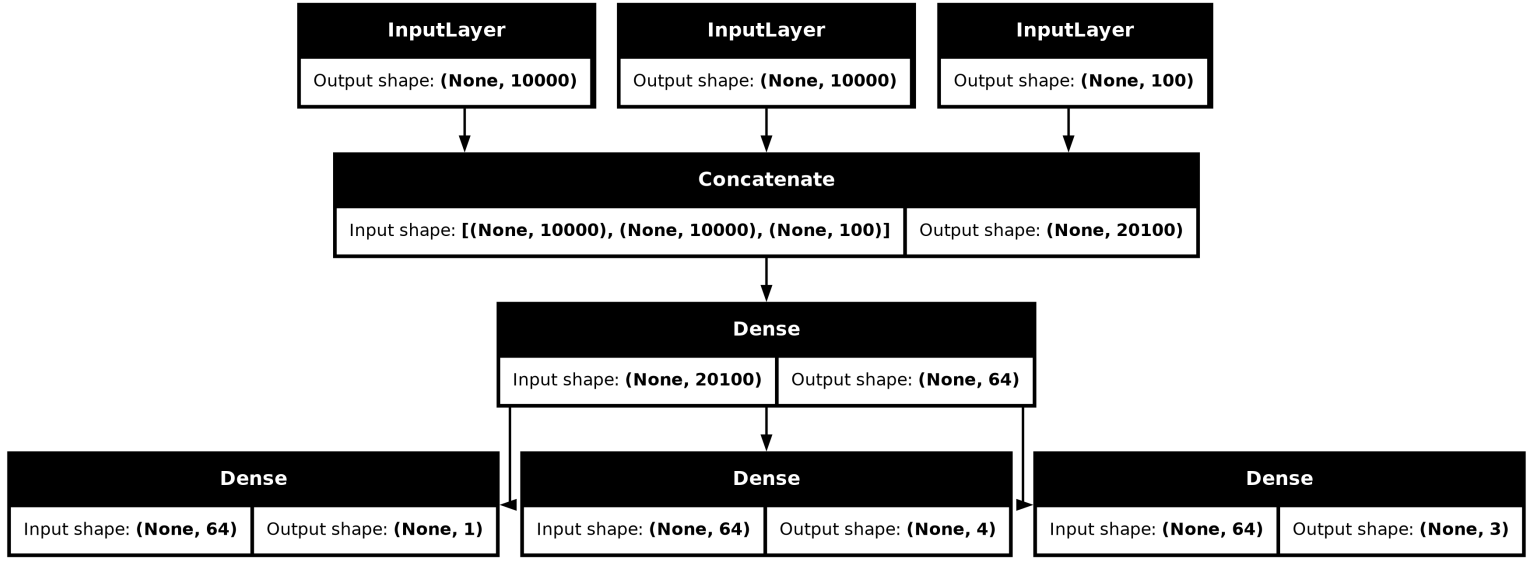
층별 입력값/출력값 정보

```
>>> model.layers[3].input
[<KerasTensor: shape=(None, 10000) dtype=float32 (created by layer 'title')>,
 <KerasTensor: shape=(None, 10000) dtype=float32 (created by layer
'text_body')>,
 <KerasTensor: shape=(None, 100) dtype=float32 (created by layer 'tags')>]

>>> model.layers[3].output
<KerasTensor: shape=(None, 20100) dtype=float32 (created by layer
'concatenate')>
```

출력층을 제외한 층 재활용

```
features = model.layers[4].output
difficulty = layers.Dense(3, activation="softmax", name="difficulty")(features)

new_model = keras.Model(
    inputs=[title, text_body, tags],
    outputs=[priority, department, difficulty])
```

```
>>> keras.utils.plot_model(new_model, "updated_ticket_classifier.png",
show_shapes=True)
```



신경망 모델 구성법 3: 서브클래싱

- `keras.Model` 클래스를 상속하는 모델 클래스를 직접 선언
- `__init__()` 메서드(생성자): 은닉층과 출력층으로 사용될 층 객체 지정
- `call()` 메서드: 층을 연결하는 과정 지정. 즉, 입력값으로부터 출력값을 만들어내는 순전파 과정 묘사.

예제: 고객 요구사항 처리 모델

```
class CustomerTicketModel(keras.Model):
    def __init__(self, num_departments):
        super().__init__()
        self.concat_layer = layers.Concatenate()
        self.mixing_layer = layers.Dense(64, activation="relu")
        self.priority_scorer = layers.Dense(1, activation="sigmoid")
        self.department_classifier = layers.Dense(
            num_departments, activation="softmax")

    def call(self, inputs):
        title = inputs["title"]
        text_body = inputs["text_body"]
        tags = inputs["tags"]

        features = self.concat_layer([title, text_body, tags])
        features = self.mixing_layer(features)
        priority = self.priority_scorer(features)
        department = self.department_classifier(features)
        return priority, department

model = CustomerTicketModel(num_departments=4)
```

서브클래싱 기법의 장단점

- 장점
    - `call()` 함수를 이용하여 층을 임의로 구성할 수 있다.
    - `for` 반복문 등 파이썬 프로그래밍 모든 기법을 적용할 수 있다.

- 단점
    - 모델 구성을 전적으로 책임져야 한다.
    - 모델 구성 정보가 `call()` 함수 외부로 노출되지 않아서 앞서 보았던 그래프 표현을 사용할 수 없다.

혼합 신경망 모델 구성법

모델은 층의 자식 클래스

- `keras.Model`이 `keras.layers.Layer`의 자식 클래스
- 모델 클래스: 모델에 포함된 층의 가중치 행렬과 편향 벡터를 활용하는 훈련, 평가, 예측을 총괄 `fit()`, `evaluate()`, `predict()` 메서드를 함께 지원할 뿐.

예제: 서브클래싱 모델을 함수형 모델에 활용하기

```
class Classifier(keras.Model):

    def __init__(self, num_classes=2):
        super().__init__()
        if num_classes == 2:
            num_units = 1
            activation = "sigmoid"
        else:
            num_units = num_classes
            activation = "softmax"
        self.dense = layers.Dense(num_units, activation=activation)

    def call(self, inputs):
        return self.dense(inputs)
```

```
inputs = keras.Input(shape=(3,))                          # 입력층
features = layers.Dense(64, activation="relu")(inputs) # 은닉층
outputs = Classifier(num_classes=10)(features)          # 출력층

model = keras.Model(inputs=inputs, outputs=outputs)
```

예제: 함수형 모델을 서브클래싱 모델에 활용하기

```
inputs = keras.Input(shape=(64,))
outputs = layers.Dense(1, activation="sigmoid")(inputs)
binary_classifier = keras.Model(inputs=inputs, outputs=outputs)
```

```
class MyModel(keras.Model):

    def __init__(self, num_classes=2):
        super().__init__()
        self.dense = layers.Dense(64, activation="relu")
        self.classifier = binary_classifier

    def call(self, inputs):
        features = self.dense(inputs)
        return self.classifier(features)
```

신경망 모델의 구성, 훈련, 평가, 예측

- 딥러닝 신경망 모델의 훈련은 한 번 시작되면 훈련이 종료될 때까지 어떤 간섭도 받지 않는다.
- 다만, 훈련 진행과정을 관찰monitoring할 수 있을 뿐이다.
- 훈련 과정 동안 관찰할 수 있는 내용은 일반적으로 다음과 같다.
    - 에포크별 손실값
    - 에포크별 평가지표

콜백

- 훈련 기록 작성
    - 훈련 에포크마다 보여지는 손실값, 평가지표 등 관리
    - `keras.callbacks.CSVLogger` 클래스 활용.
- 훈련중인 모델의 상태 저장
    - 훈련 중 가장 좋은 성능의 모델(의 상태) 저장
    - `keras.callbacks.ModelCheckpoint` 클래스 활용
- 훈련 조기 종료
    - 검증셋에 대한 손실이 더 이상 개선되지 않는 경우 훈련을 종료 시키기
    - `keras.callbacks.EarlyStopping` 클래스 활용
- 하이퍼 파라미터 조정
    - 학습률 동적 변경 지원
    - `keras.callbacks.LearningRateScheduler` 또는 `keras.callbacks.ReduceLROnPlateau` 클래스 활용

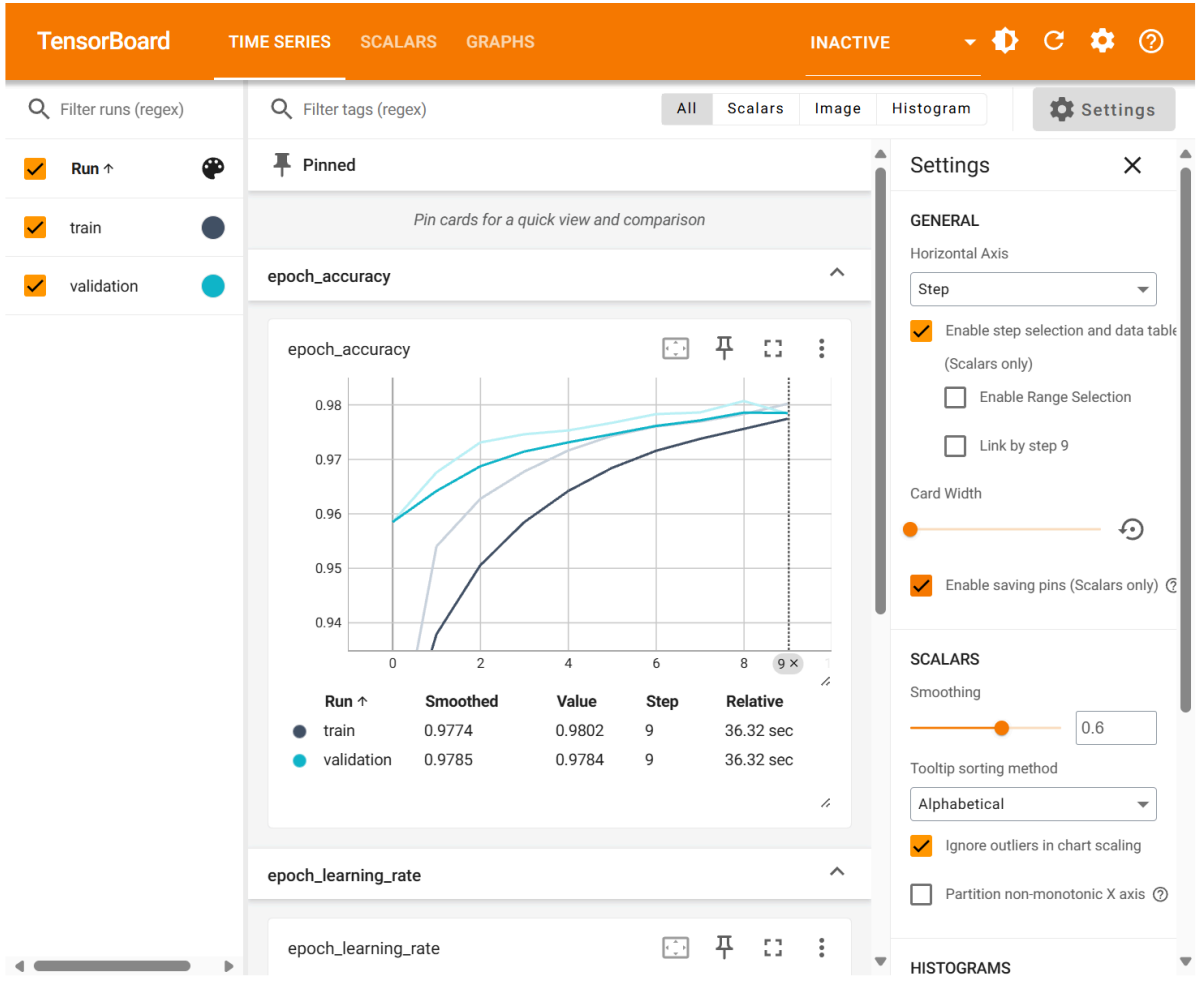
예제

```
callbacks_list = [
    keras.callbacks.EarlyStopping(
        monitor="val_accuracy",
        patience=2,
    ),
    keras.callbacks.ModelCheckpoint(
        filepath="checkpoint_path",
        monitor="val_loss",
        save_best_only=True,
    )
]
```

```
def get_mnist_model():
    inputs = keras.Input(shape=(28 * 28,))
    features = layers.Dense(512, activation="relu")(inputs)
    features = layers.Dropout(0.5)(features)
    outputs = layers.Dense(10, activation="softmax")(features)
    model = keras.Model(inputs, outputs)
    return model

model = get_mnist_model()

model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

model.fit(train_images, train_labels,
          epochs=10,
          callbacks=callbacks_list,
          validation_data=(val_images, val_labels))
```

텐서보드

- 신경망 모델 구조 시각화
- 손실값, 정확도 등의 변화 시각화
- 가중치, 편향 텐서 등의 변화 히스토그램
- 이미지, 텍스트, 오디오 데이터 시각화
- 기타 다양한 기능 제공


TensorBoard
TIME SERIES
SCALARS
GRAPHS
INACTIVE
Filter runs (regex)
Filter tags (regex)
All
Scalars
Image
Histogram
Settings
Run
train
validation
Pinned
Pin cards for a quick view and comparison
epoch_accuracy
epoch_accuracy
0.98
0.97
0.96
0.95
0.94
0
2
4
6
8
9
Run
Smoothed
Value
Step
Relative
train
0.9774
0.9802
9
36.32 sec
validation
0.9785
0.9784
9
36.32 sec
epoch_learning_rate
epoch_learning_rate
Settings
GENERAL
Horizontal Axis
Step
Enable step selection and data table
(Scalars only)
Enable Range Selection
Link by step 9
Card Width
Enable saving pins (Scalars only)
SCALARS
Smoothing
0.6
Tooltip sorting method
Alphabetical
Ignore outliers in chart scaling
Partition non-monotonic X axis
HISTOGRAMS

텐서보드는 `TensorBoard` 콜백 클래스를 활용한다.

- `log_dir` : 텐서보드 서버 실행에 필요한 데이터 저장소 지정

```
tensorboard = keras.callbacks.TensorBoard(
    log_dir="./tensorboard_log_dir",
)

model.fit(train_images, train_labels,
          epochs=10,
          validation_data=(val_images, val_labels),
          callbacks=[tensorboard])
```

- 주피터 노트북에서

```
%load_ext tensorboard
%tensorboard --logdir ./tensorboard_log_dir
```

- 터미널에서

```
$ tensorboard --logdir ./tensorboard_log_dir
```