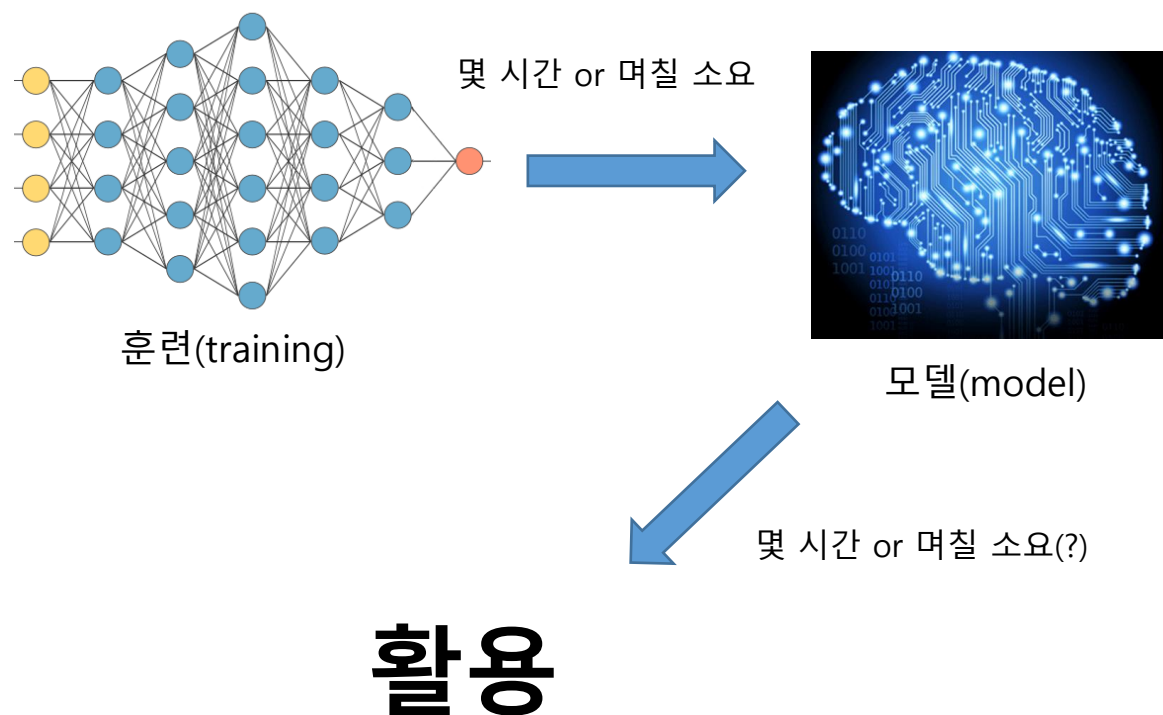


인공지능(AI)

05주차. 딥러닝 모델과 훈련 다루기

비유 내용

딥러닝 모델의 저장과 활용

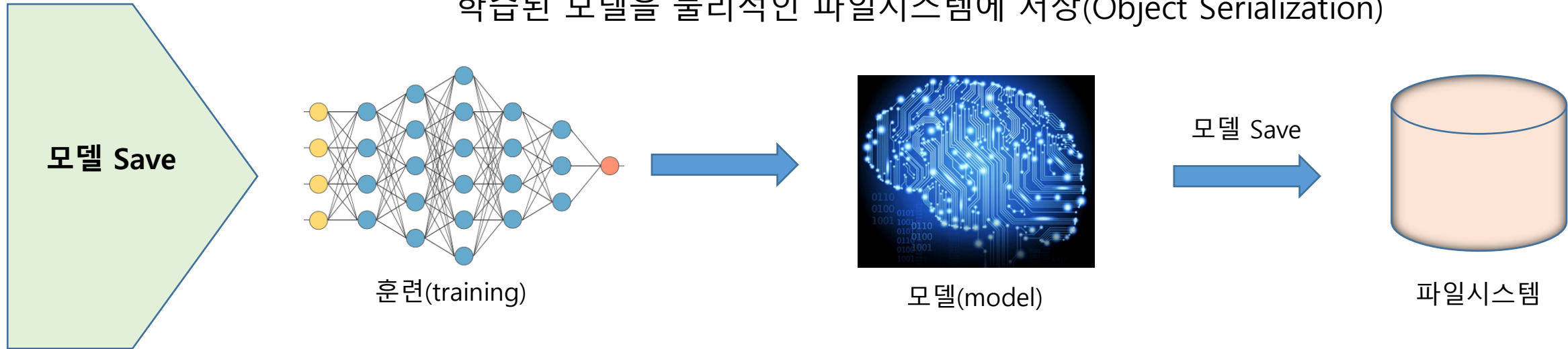


몇 시간 (혹은 며칠) 동안 딥러닝 모델을 학습 시킨 후 만족할만한 결과 모델을 얻어 실무에 바로 적용시키려고 한다. 이 때 떠오르는 의문 중 하나가 "딥러닝 모델을 사용하려면 매번 이렇게 몇 시간(혹은 며칠) 동안 학습시켜야 되는 걸까?"

절대 그렇지 않다.

딥러닝 모델을 학습시킨다는 것은 딥러닝 모델을 구성하는 노드들에 적용할 최적의 가중치(weight)를 찾는 과정이다. 설계 모델의 구성과 노드들에 적용할 가중치만 저장해 놓으면, 필요할 때 마다 저장한 설계 모델 구성과 가중치를 로드하여 사용하면 된다.

학습된 모델을 물리적인 파일시스템에 저장(Object Serialization)



물리적인 파일시스템에 저장된 모델을 복원하여 예측에 활용(Object Deserialization)



모델 save를 위한 디렉토리 생성

```
In [32]: import os

if not os.path.exists('./model'):
    os.mkdir('model')
```

model.h5 파일로 모델 save

```
In [33]: model.save('model/model.h5')
```

save된 모델 확인

이름	수정한 날짜	유형	크기
 model.h5	2019-08-25 오후 8:57	H5 파일	417KB

모델 restore

```
In [2]: from keras.models import load_model  
        model = load_model('model/model.h5')  
        Using TensorFlow backend.
```

restore 된 모델 확인

```
In [3]: model.summary()
```

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	401920
dense_2 (Dense)	(None, 10)	5130

Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0

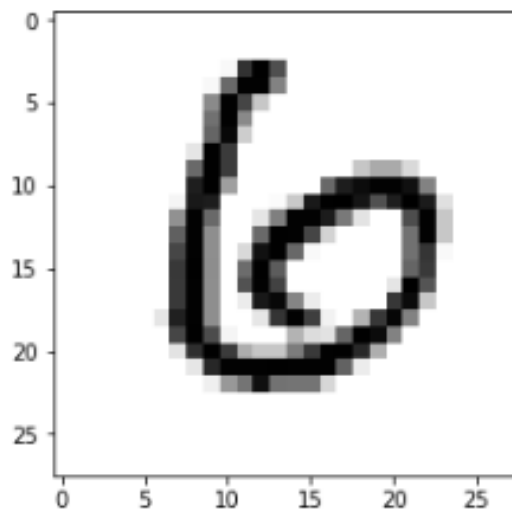
prediction을 위한 mnist 이미지 로드

```
In [4]: from keras.datasets import mnist
import matplotlib.pyplot as plt

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

```
In [5]: pick_img = test_images[11]
```

```
In [6]: plt.imshow(pick_img, cmap=plt.cm.binary)
plt.show()
```



restore된 모델을 이용한 mnist 이미지 prediction

```
In [7]: pick_img.shape
```

```
Out[7]: (28, 28)
```

```
In [8]: new_img = pick_img.reshape(1, 784, )
```

```
In [9]: result = model.predict_classes(new_img)
```

```
In [10]: result[0]
```

```
Out[10]: 6
```

모델 훈련(training)

```
In [23]: model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Epoch 1/5  
60000/60000 [=====] - 6s 93us/step - loss: 0.2208 - acc: 0.9349  
Epoch 2/5  
60000/60000 [=====] - 3s 42us/step - loss: 0.0919 - acc: 0.9722  
Epoch 3/5  
60000/60000 [=====] - 3s 42us/step - loss: 0.0626 - acc: 0.9819  
Epoch 4/5  
60000/60000 [=====] - 3s 42us/step - loss: 0.0626 - acc: 0.9819  
Epoch 5/5  
60000/60000 [=====] - 3s 42us/step - loss: 0.0626 - acc: 0.9819
```

```
Out [23]: <keras.callbacks.History object at 0x0000000000000000>
```

model_save.ipynb 실습

모델 평가

```
In [24]: test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
10000/10000 [=====]
```

```
In [25]: print('test_acc:', test_acc)
```

```
test_acc: 0.9783
```

model_restore.ipynb 실습

모델 save

```
In [32]: import os
```

```
if not os.path.exists('./model'):  
    os.mkdir('model')
```

```
In [33]: model.save('model/model.h5')
```


TensorBoard 개요

TensorFlow에서 기록한 로그를 이용하여 TensorFlow의 최적화를 지원하기 위한 시각화 도구
backend로 TensorFlow를 사용하는 경우 Keras에서도 사용 가능

TensorBoard 사용방법

- ① 소스코드에서 TensorBoard를 사용하기 위한 코드 작성
- ② Windows command 창에서 TensorBoard 기동
- ③ 브라우저에서 TensorBoard 접속 및 사용

① 소스코드에서 Tensorboard를 사용하기 위한 코드 작성

TensorBoard import 및 TensorBoard 객체 생성

```
In [3]: from keras.callbacks import TensorBoard  
tensorboard = TensorBoard(log_dir="d:/temp/logs")
```

모델 훈련을 위한 fit 함수 호출 시 callback으로 tensorboard 객체 설정

```
In [12]: model.fit(train_images, train_labels, epochs=5, batch_size=64,  
                  callbacks=[tensorboard])
```

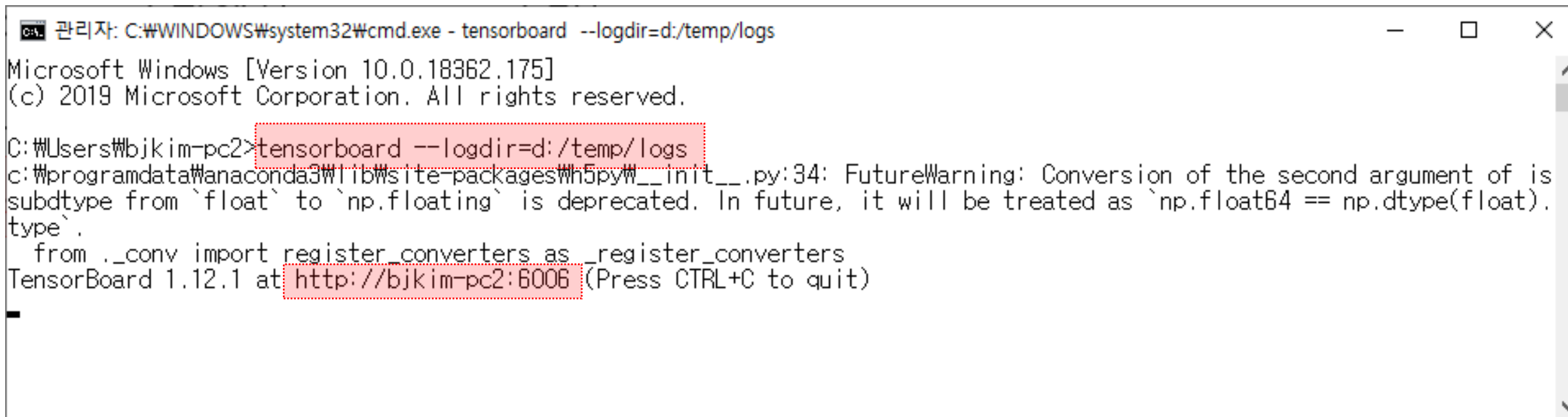
해당 소스 실행 후 TensorBoard Event 파일 생성 확인

이름	수정한 날짜	유형	크기
events.out.tfevents.1566734215.BJKIM-...	2019-08-25 오후 8:57	BJKIM-PC2 파일	726KB

② Windows command 창에서 TensorBoard 기동

키보드에서 윈도우키 + R, 실행 창에서 cmd 입력 후 엔터

```
tensorboard --logdir=d:/temp/logs
```

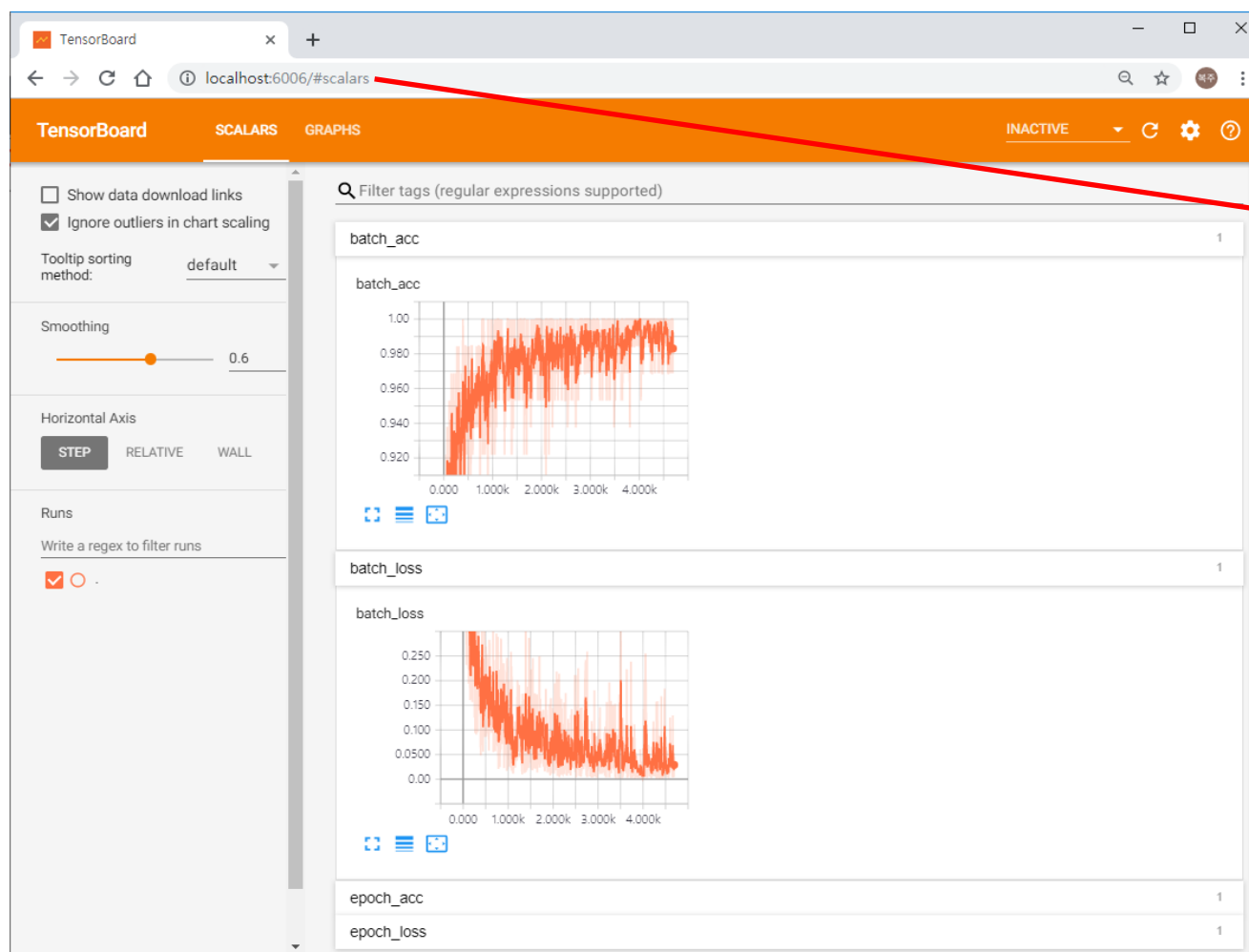


```
관리자: C:\WINDOWS\system32\cmd.exe - tensorboard --logdir=d:/temp/logs
Microsoft Windows [Version 10.0.18362.175]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\bjkim-pc2>tensorboard --logdir=d:/temp/logs
c:\programdata\anaconda3\lib\site-packages\h5py\__init__.py:34: FutureWarning: Conversion of the second argument of is
subtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).
type`.
  from ._conv import register_converters as _register_converters
TensorBoard 1.12.1 at http://bjkim-pc2:6006 (Press CTRL+C to quit)
```

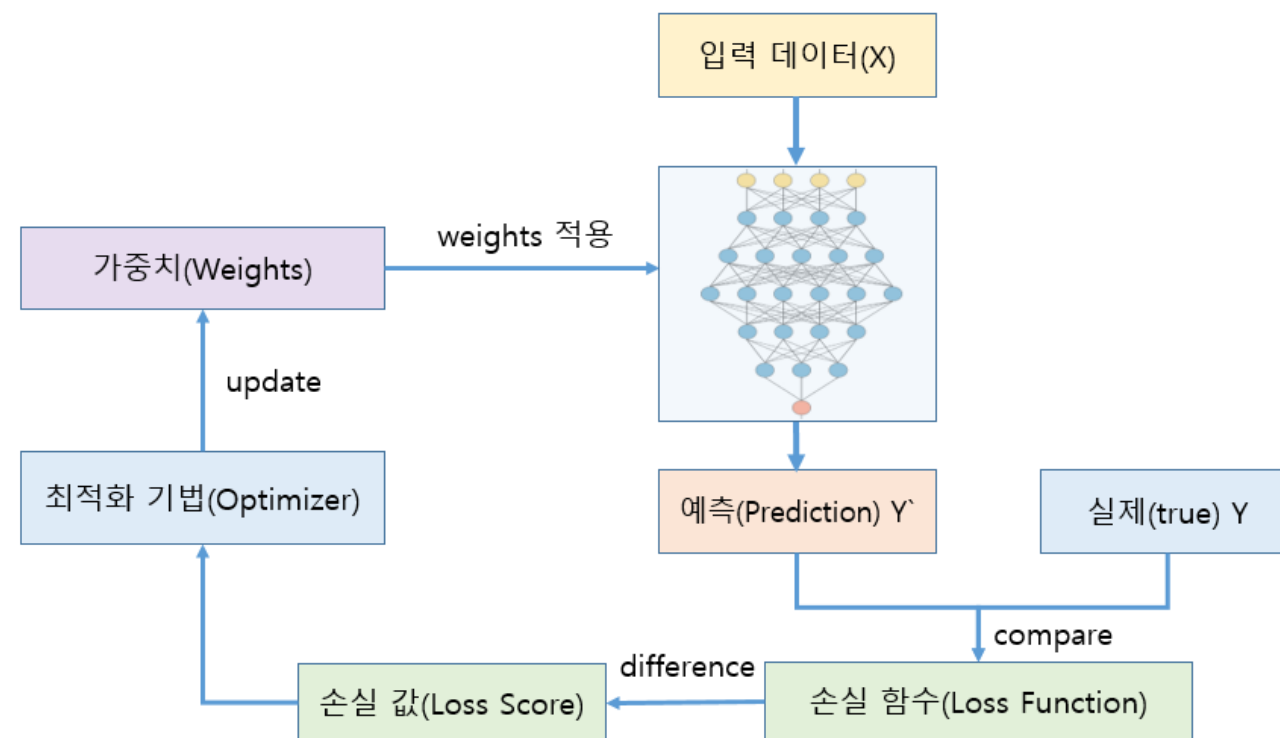
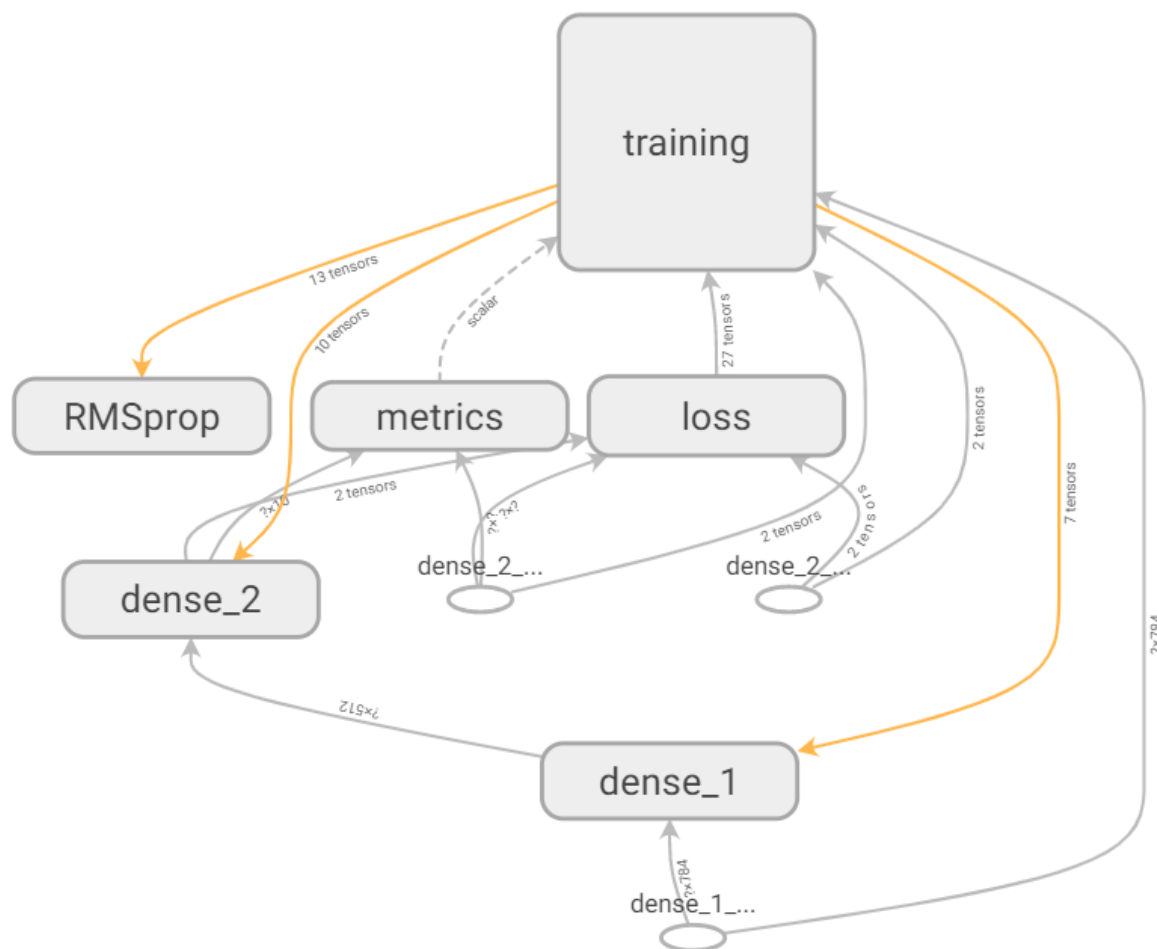
③ 브라우저에서 TensorBoard 접속 및 사용

크롬(chrome) 브라우저 기동 후 주소 입력란에 `http://localhost:6006`



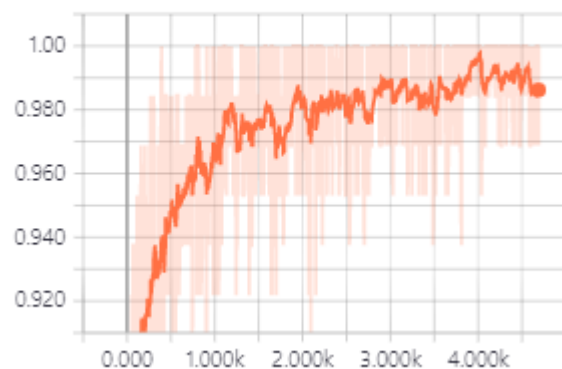
`http://localhost:6006`

TensorBoard의 GRAPHS 메뉴에서 모델 확인 가능

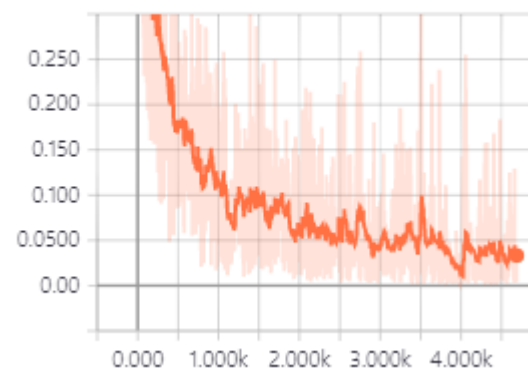


TensorBoard를 이용하여 학습진행 과정의 batch와 epoch의 acc(accuracy)와 loss 확인 가능

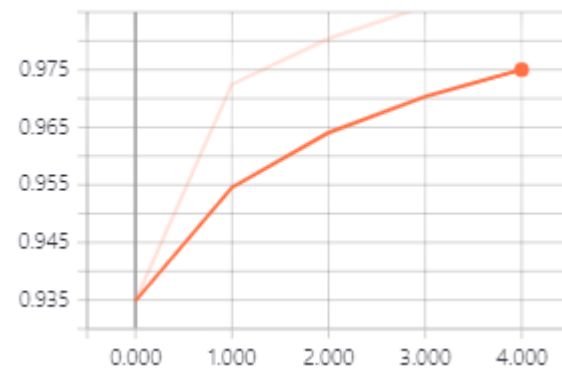
batch_acc



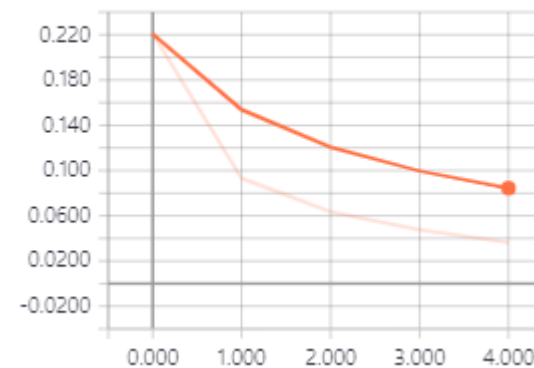
batch_loss



epoch_acc



epoch_loss



네트워크 모델 설계

```
In [7]: from keras import models
        from keras import layers
```

```
In [8]: model = models.Sequential()
        model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
        model.add(layers.Dense(10, activation='softmax'))
```

```
In [9]: from tensorflow.python.keras.callbacks import TensorBoard
```

```
tensorboard = TensorBoard(log_dir='./logs')
# Create logs directory if not exists
os.makedirs(log_dir, exist_ok=True)
```

```
In [10]: model
```

```
In [11]: model
```

```
-----
Layer (type)                 Output Shape          Param #
-----
dense_1 (Dense)              (None, 512)           401920
dense_2 (Dense)              (None, 10)            5130
-----
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
-----
```

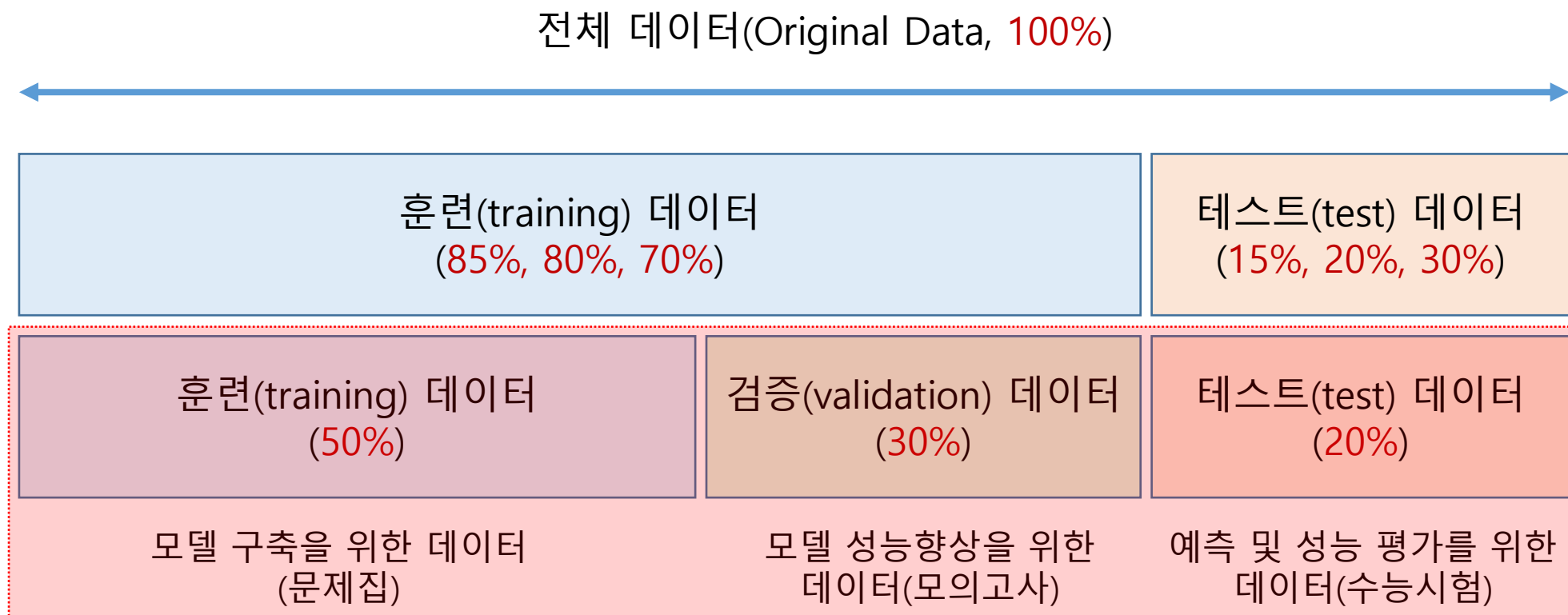
tensorboard.ipynb 실습

모델 훈련(training)

```
In [12]: model.fit(train_images, train_labels, epochs=5, batch_size=64,
                  callbacks=[tensorboard])
```

```
Epoch 1/5
60000/60000 [=====] - 30s 500us/step - loss: 0.2208 - acc: 0.9349
Epoch 2/5
60000/60000 [=====] - 35s 591us/step - loss: 0.0930 - acc: 0.9725
Epoch 3/5
60000/60000 [=====] - 35s 576us/step - loss: 0.0636 - acc: 0.9804
Epoch 4/5
60000/60000 [=====] - 34s 560us/step - loss: 0.0474 - acc: 0.98603s - loss - ETA:
_
```

머신러닝/딥러닝 학습데이터는 훈련(training) 데이터, 검증(validation) 데이터 및 테스트(test)로 분할(split)하여 사용



훈련 데이터 : 모델의 훈련 및 가중치 업데이트 등의 목적으로 사용

검증 데이터 : 훈련된 모델의 평가 및 최종 모델을 선정하기 위해 사용

테스트 데이터 : 모델의 예측 및 평가를 위해 사용

훈련 시 검증 데이터를 사용하기 위한 설정 `validation_split` 옵션 파라미터 이용

훈련 과정의 검증 데이터 손실 값(`val_loss`)과 정확도(`val_accuracy`) 출력

```
model.fit(train_images, train_labels, epochs=5, batch_size=64,  
          validation_split = 0.2)
```

WARNING:tensorflow:From C:\Users\bjkim\Anaconda3\envs\tf_env\lib\site-packages\keras\backend\tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.

Train on 40000 samples, validate on 10000 samples

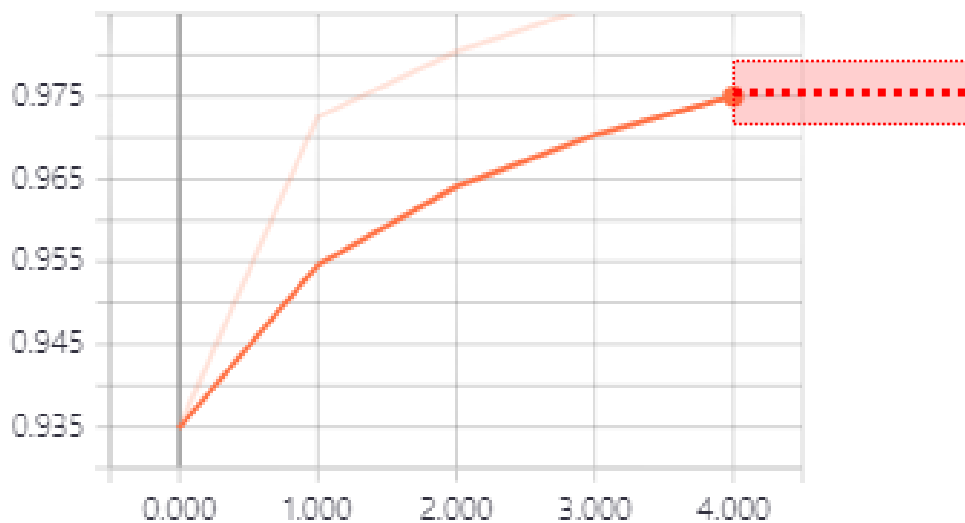
Epoch 1/5	40000/40000 [=====]	- 2s 44us/step	- loss: 0.3859	- accuracy: 0.8946	- val_loss: 0.2600	- val_accuracy: 0.9242
Epoch 2/5	40000/40000 [=====]	- 1s 36us/step	- loss: 0.2028	- accuracy: 0.9405	- val_loss: 0.2065	- val_accuracy: 0.9402
Epoch 3/5	40000/40000 [=====]	- 1s 36us/step	- loss: 0.1570	- accuracy: 0.9538	- val_loss: 0.1958	- val_accuracy: 0.9400
Epoch 4/5	40000/40000 [=====]	- 1s 36us/step	- loss: 0.1278	- accuracy: 0.9630	- val_loss: 0.1560	- val_accuracy: 0.9527
Epoch 5/5	40000/40000 [=====]	- 1s 35us/step	- loss: 0.1083	- accuracy: 0.9686	- val_loss: 0.1539	- val_accuracy: 0.9555

학습 조기종료(Early Stopping)

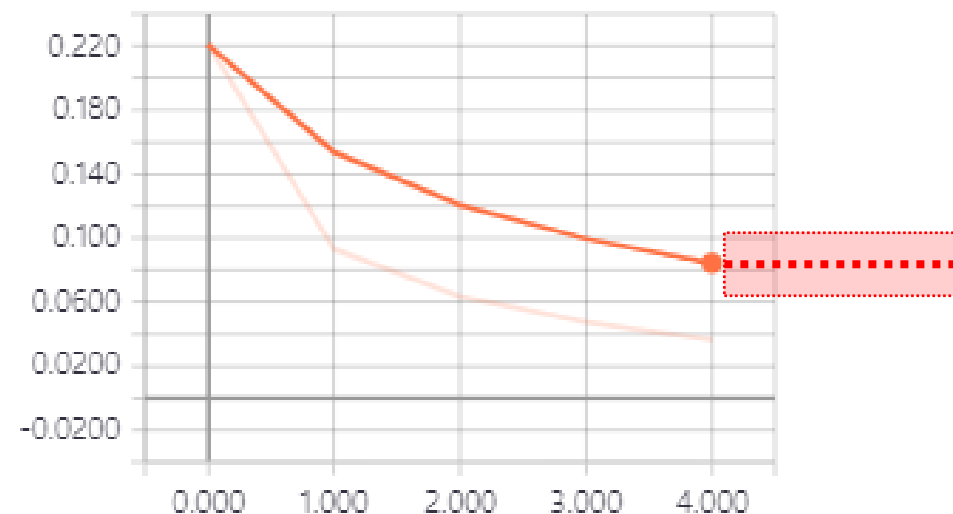
학습 진행 과정에서 반복학습의 효과가 없을 때 학습을 종료

정확도(acc)가 증가하지 않고, 손실 값(loss)이 감소하지 않는 학습 정체구간 진입 시 학습 종료
반복학습의 효과가 없는 경우는 Hyper Parameter 튜닝

epoch_acc



epoch_loss



EarlyStopping import 및 EarlyStopping 객체 생성

```
In [68]: from keras.callbacks import EarlyStopping  
early_stopping = EarlyStopping()
```

- monitor : 학습 종료 판단 지표, 'val_loss'나 'val_acc'가 주로 사용
- patience : 학습효과가 없다고 판단하더라도 즉시 종료하지 않고,
효과가 없는 epoch를 얼마나 기다릴 것인가 지정,
(3으로 지정하게 되면 학습효과가 없는 epoch가 3번째 지속될 경우 학습 종료)

모델 훈련 시 callbacks 옵션 파라미터에 EarlyStopping적용

```
model.fit(train_images, train_labels, epochs=100, batch_size=64,  
          validation_split=0.2,  
          callbacks=[tensorboard, early_stopping])
```

조기 학습종료(Early Stopping)를 적용하지 않는 경우

```
In [13]: model.fit(train_images, train_labels, epochs=100, batch_size=64,
                  validation_data=(val_images, val_labels))
```

Train on 50000 samples, validate on 10000 samples

```
Epoch 1/100
50000/50000 [=====] - 4s 86us/step - loss: 0.3585 - acc: 0.9019 - val_loss: 0.2073 - val_acc: 0.9413
Epoch 2/100
50000/50000 [=====] - 2s 46us/step - loss: 0.1808 - acc: 0.9471 - val_loss: 0.1527 - val_acc: 0.9554
Epoch 3/100
50000/50000 [=====] - 2s 46us/step - loss: 0.1363 - acc: 0.9602 - val_loss: 0.1242 - val_acc: 0.9646
Epoch 4/100
50000/50000 [=====] - 2s 46us/step - loss: 0.1099 - acc: 0.9675 - val_loss: 0.1194 - val_acc: 0.9651
Epoch 5/100
50000/50000 [=====] - 2s 47us/step - loss: 0.0928 - acc: 0.9728 - val_loss: 0.1107 - val_acc: 0.9689
Epoch 6/100
50000/50000 [=====] - 2s 47us/step - loss: 0.0808 - acc: 0.9759 - val_loss: 0.0986 - val_acc: 0.9711
Epoch 7/100
50000/50000 [=====] - 2s 49us/step - loss: 0.0712 - acc: 0.9785 - val_loss: 0.0998 - val_acc: 0.9716
Epoch 8/100
50000/50000 [=====] - 2s 49us/step - loss: 0.0647 - acc: 0.9815 - val_loss: 0.1048 - val_acc: 0.9702
Epoch 9/100
50000/50000 [=====] - 2s 47us/step - loss: 0.0293 - acc: 0.9918 - val_loss: 0.1002 - val_acc: 0.9764
Epoch 18/100
50000/50000 [=====] - 2s 46us/step - loss: 0.0273 - acc: 0.9926 - val_loss: 0.1049 - val_acc: 0.9745
Epoch 19/100
50000/50000 [=====] - 2s 46us/step - loss: 0.0255 - acc: 0.9931 - val_loss: 0.1080 - val_acc: 0.9756
Epoch 20/100
50000/50000 [=====] - 2s 49us/step - loss: 0.0231 - acc: 0.9936 - val_loss: 0.1139 - val_acc: 0.9736
Epoch 21/100
50000/50000 [=====] - 3s 51us/step - loss: 0.0214 - acc: 0.9941 - val_loss: 0.1211 - val_acc: 0.9741
Epoch 22/100
50000/50000 [=====] - 2s 49us/step - loss: 0.0201 - acc: 0.9946 - val_loss: 0.1162 - val_acc: 0.9753
Epoch 23/100
50000/50000 [=====] - 2s 46us/step - loss: 0.0190 - acc: 0.9954 - val_loss: 0.1143 - val_acc: 0.9746
Epoch 24/100
50000/50000 [=====] - 2s 48us/step - loss: 0.0181 - acc: 0.9955 - val_loss: 0.1261 - val_acc: 0.9741
Epoch 25/100
20096/50000 [=====>.....] - ETA: 1s - loss: 0.0161 - acc: 0.9956
```

학습효과가 없음에도
epochs 100 까지
반복학습 진행

조기 학습종료(Early Stopping)를 적용한 경우

```
In [14]: model.fit(train_images, train_labels, epochs=100, batch_size=64,
                  validation_data=(val_images, val_labels),
                  callbacks=[tensorboard, early_stopping])
```

Train on 50000 samples, validate on 10000 samples

Epoch	50000/50000	Time	Step	loss	acc	val_loss	val_acc
Epoch 1/100	[=====]	3s	54us/step	0.3576	0.9006	0.1942	0.9464
Epoch 2/100	[=====]	2s	48us/step	0.1840	0.9455	0.1535	0.9561
Epoch 3/100	[=====]	2s	46us/step	0.1368	0.9592	0.1263	0.9635
Epoch 4/100	[=====]	2s	47us/step	0.1097	0.9675	0.1167	0.9648
Epoch 5/100	[=====]	2s	46us/step	0.0925	0.9725	0.1021	0.9703
Epoch 6/100	[=====]	2s	47us/step	0.0806	0.9769	0.0967	0.9712
Epoch 7/100	[=====]	2s	47us/step	0.0710	0.9794	0.0969	0.9728

Out [14]: <keras.callbacks.History at 0x23e0230af98>

Optimizer가 학습효과가 없음을 판단하여 학습 조기 종료

Early Stopping 미적용

```
In [89]: # model.fit(train_images, train_labels, epochs=100, batch_size=64,  
#           validation_data=(val_images, val_labels))
```

Early Stopping 적용 / patience 미적용

```
In [90]: model.fi
```

```
Train on 50000/50000  
Epoch 1/50000/50000  
Epoch 2/50000/50000  
Epoch 3/50000/50000  
Epoch 4/50000/50000  
Epoch 5/50000/50000  
Epoch 6/100  
50000/50000 [=====] - 2s 40us/step - loss: 0.0900 - acc: 0.9714 - val_loss: 0.1102 - val_acc: 0.9680  
Epoch 6/100  
50000/50000 [=====] - 2s 48us/step - loss: 0.0862 - acc: 0.9748 - val_loss: 0.1108 - val_acc: 0.9672
```

```
Out [90]: <keras.callbacks.History at 0x23e05a8e668>
```

Early Stopping 적용 / patience 적용

```
In [91]: # early_stopping = EarlyStopping(patience = 10)  
# model.fit(train_images, train_labels, epochs=100, batch_size=64,  
#           validation_data=(val_images, val_labels),  
#           callbacks=[tensorboard, early_stopping])
```

validation.ipynb 실습

earlystopping.ipynb 실습

형성평가

번호	문제	보기	정답	해설
1	훈련된 딥러닝 모델을 시스템에 저장(save) 후 복원(restore)하여 예측(prediction)에 활용할 수 있다.		○	학습된 모델을 물리적인 파일시스템에 저장(Object Serialization)한 후 물리적인 파일시스템에 저장된 모델을 복원(Object erialization)하여 예측에 활용할 수 있다.
2	다음 중 Keras에서 모델을 저장하기 위한 함수와 복원을 위한 함수로 올바르게 연결된 것은?	① save(), read() ② write(), read() ③ save(), restore() ④ write(), restore()	②	Keras에서 모델을 저장하기 위한 함수는 save() 이고, 복원을 위한 함수는 restore() 이다.

형성평가

번호	문제	보기	정답	해설
3	TensorFlow에서 기록한 로그를 이용하여 TensorFlow의 최적화를 지원하기 위한 시각화 도구로 backend로 TensorFlow를 사용하는 경우 Keras에서도 사용 가능한 시각화 도구는 무엇인가?	<div>(ㄱ)</div>	TensorBoard	TensorBoard는 TensorFlow에서 기록한 로그를 이용하여 TensorFlow의 최적화를 지원하기 위한 시각화 도구로 backend로 TensorFlow를 사용하는 경우 Keras에서도 사용 가능한 시각화 도구이다.

학습정리

- 유닛 1

모델 save는 학습된 모델을 물리적인 파일시스템에 저장(Object Serialization)

모델 restore는 물리적인 파일시스템에 저장된 모델을 복원하여 예측에 활용(Object erialization)

모델 save를 위한 디렉토리 생성 후 model.save() 함수를 사용하여 h5 파일로 모델 저장

모델 restore를 위해 model.restore() 함수 사용

- 유닛 2

TensorBoard는 TensorFlow에서 기록한 로그를 이용하여 TensorFlow의 최적화를 지원하기 위한 시각화 도구로 backend로 TensorFlow를 사용하는 경우 Keras에서도 사용 가능

TensorBoard를 사용하는 방법은 ① 소스코드에서 TensorBoard를 사용하기 위한 코드 작성 ② Windows command 창에서 TensorBoard 기동

③ 브라우저에서 TensorBoard 접속 및 사용

학습정리

- 유닛 3

머신러닝/딥러닝 학습데이터는 훈련(training) 데이터, 검증(validation) 데이터 및 테스트(test)로 분할(split)하여 사용

검증(Validation) 데이터 사용을 위해 이미지/레이블 분할하여 사용

훈련 시 검증 데이터를 사용하기 위해 model.fit함수에 validation_data 파라미터를 지정

validation_data 파라미터를 지정하면 훈련 과정의 검증 데이터 손실 값(val_loss)과 정확도(val_acc) 출력

- 유닛 4

조기 학습종료(Early Stopping)은 학습 진행 과정에서 반복학습의 효과가 없을 때 학습을 종료하는 것으로 정확도(acc)가 증가하지 않고, 손실 값(loss)이 감소하지 않는 학습 정체구간 진입 시 학습 종료

EarlyStopping 적용을 위해 EarlyStopping import 및 EarlyStopping 객체 생성 후 모델 훈련 시 fit 함수의 파마미터 callbacks에 EarlyStopping적용