과목 명: 시스템프로그래밍

담당 교수 명: 김 지 환

<<Assignment 5>>

**서강대학교 컴퓨터학과**

**[20151616]**

**[최승환]**

1. 프로그램 설명

CIFAR-10 에서 제공하는 60000장의 이미지를 이용한 분류 MultiLayer Perceptron(MLP) 프로그램이다.

1. 프로그램 개요

프로그램은 다음과 같은 단계들을 거쳐 학습이 되어 평가가 시행된다.

각 단계에 대한 설명은 모듈 정의에서 하도록 하겠다.

* 1. Layer 쌓기
  2. Optimize
  3. Compile
  4. Train
  5. Evaluate

1. 모듈 정의
   1. Layer 쌓기

model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same',

input\_shape=x\_train.shape[1:]))

model.add(Dense(512, activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

* + 1. 생성한 모델을 이용해 Layer를 쌓는다. Layer 별로 BackPropagation 을 이용해 Weight 를 결정하고, Sigmoid 나 Relu 같은 함수를 이용해, 분류 시 output 을 결정한다.
  1. Optimize

optimizer = keras.optimizers.SGD(lr=0.05)

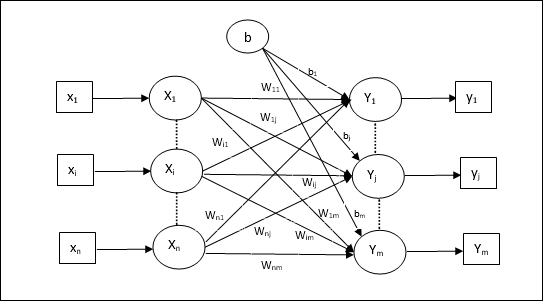
* + 1. 학습 반영 정도를 설정해준다.
  1. Compile

model.compile(loss='categorical\_crossentropy',

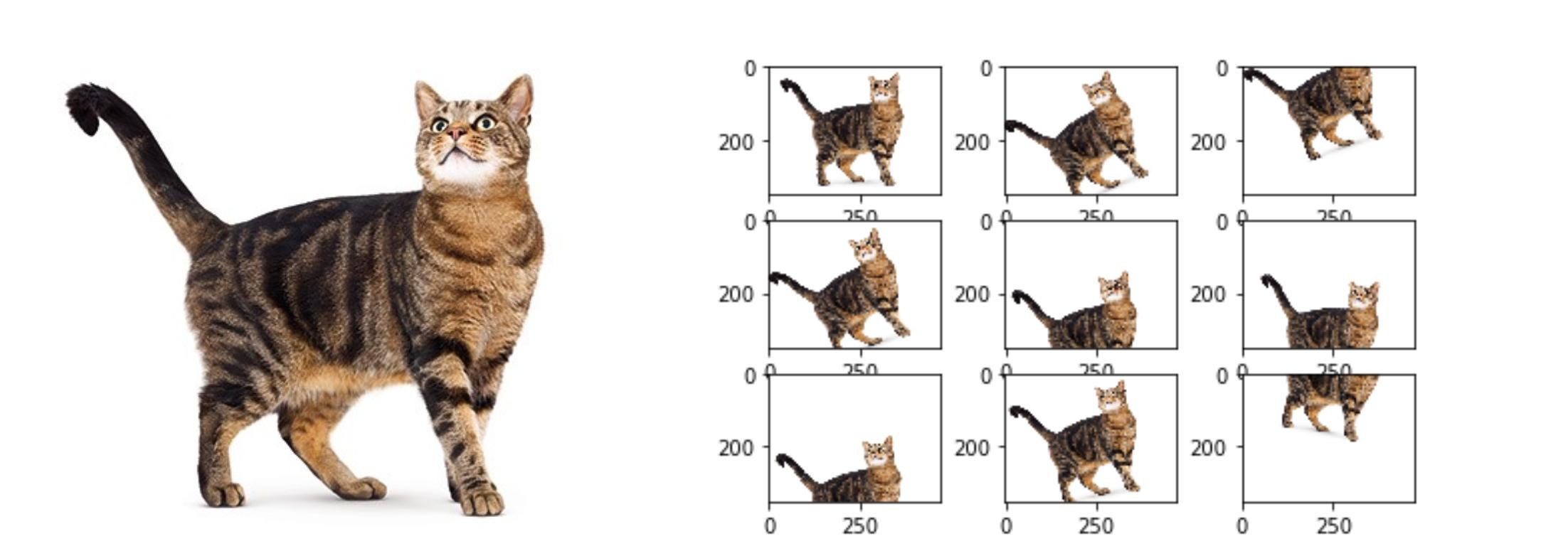
optimizer=optimizer,

metrics=['accuracy'])

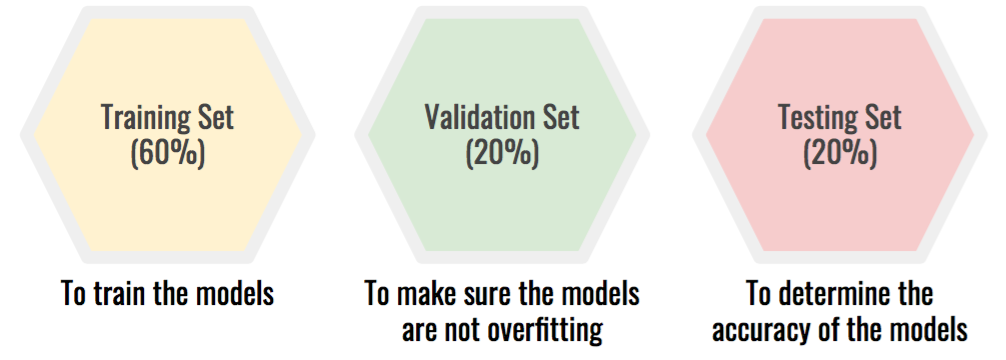
* + 1. 설정한 Layer 들을 이용해 compile 하면 학습이 가능한 모델이 된다.
  1. Train
     1. Train 을 이용해 epoch 별로 학습을 진행하며, 진행정도와 정답률을 출력해준다. 이 과정에서 back propagation, data\_augmentation 과 validation\_split 을 사용했으며, 각각은 다음과 같은 기능을 가지고 있다.
     2. Back Propagation



* + - 1. Back Propagation 은 Layer에서 다른 Layer 의 노드를 계산할 때 이용하는 기법이다. 다음 Layer 의 노드를 계산하면서, 현재 Layer 의 weight 들을 바꾸게 되는데, 이것을 Back propagation이라고 한다.
      2. Back Propagation 을 통해 학습을 진행할 수록 error 가 0에 수렴하게 된다.
    1. Data augmentation



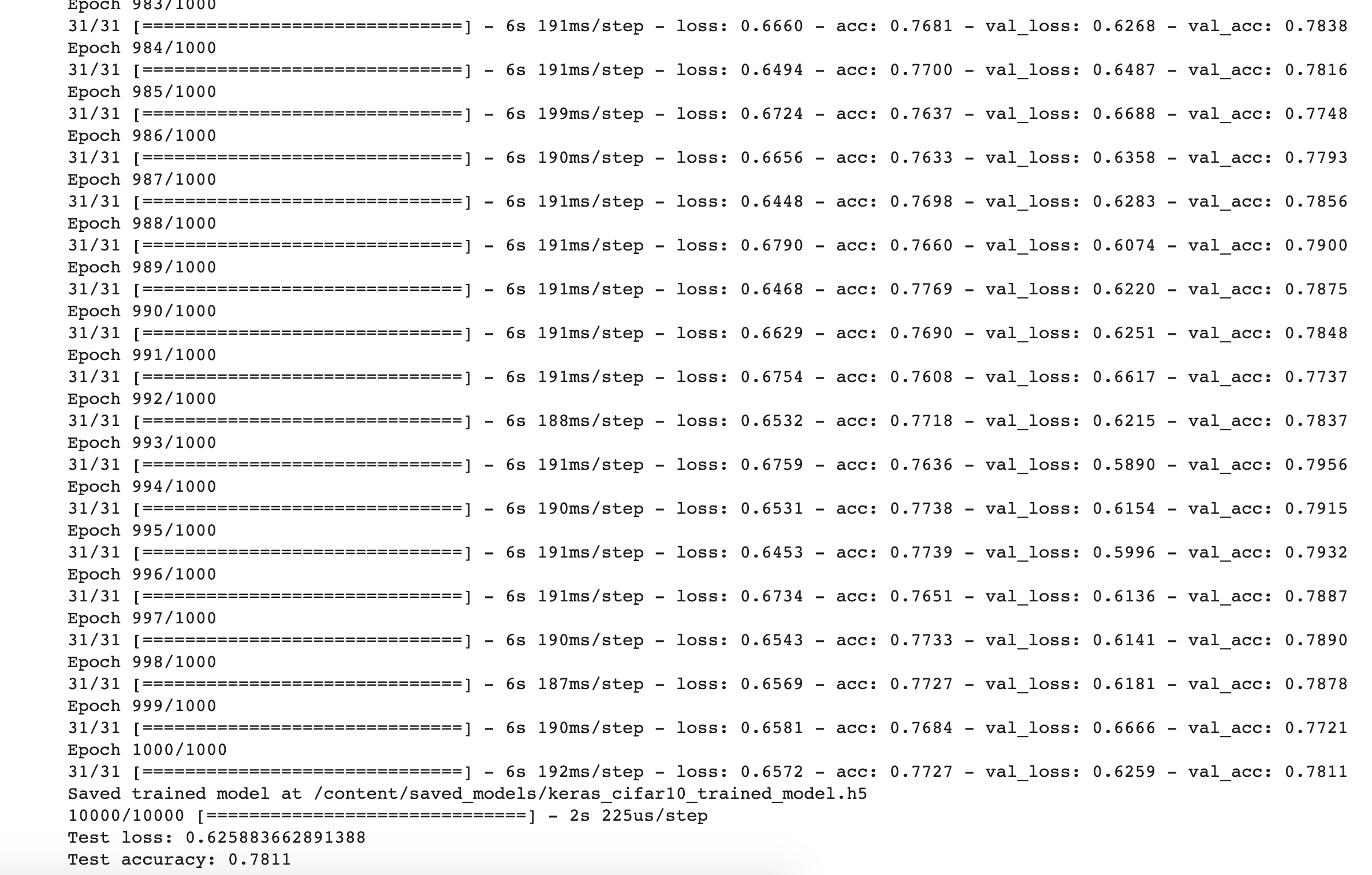
* + - 1. 입력 데이터 셋을 늘려주는 역할을 한다. 같은 이미지라고 해도, 약간의 조작을 거치면 다른 입력 데이터로 사용될 수 있다. 예를 들어, 이미지 중심의 위치를 옮긴다던지 크기를 다르게 한다던지 하는 방식으로 여러 가지 이미지를 학습시킬 수 있다.
      2. 입력 데이터 셋의 종류가 많아지면, 더 높은 정확도를 얻을 확률이 높아진다.
    1. Validation set



* + - 1. Training set 이란 학습을 위해 입력되는 데이터 셋을 의미한다. 이 경우에는 60000장의 이미지가 Training set 에 해당한다. 하지만, 모델의 학습을 마친 뒤에 이미 가지고 있는 입력을 이용해 그 모델을 평가한다면, 아무 의미가 없다. 같은 입력에 대한 결과물이기 때문이다. 이 때문에 Training set 은 Training set, Validation set 으로 나뉘어진다.
      2. Validation set 은 모델을 평가한다는 점에서 Training set 과는 다르지만, Test set 과도 다르다. Test set 은 말 그대로 모델을 평가하기 위한 데이터 집합이다. Validation set 도 평가를 하지만, 모델의 유요성을 판별하기 위해서보다는 모델에서 사용하는 인자들을 조정해주기 위해 가지고 있는 집합이다.
  1. Evaluate
     1. Test set 을 이용해 모델을 평가하고 정확도와 에러율에 대한 결과값을 얻는다.

1. 전역변수 설명
   1. batch\_size = 128 - 한 번(반복문)에 학습에 참조하는 데이터의 개수입니다.
   2. num\_classes = 10 - 분류할 클래스의 종류
   3. epochs = 10 - 학습하는 횟수
   4. data\_augmentation = True - data\_augmentation을 시행할 지
   5. steps\_per\_epoch = 1000 - 한 번의 학습에 몇 번의 과정을 거치는지
2. 결과물 및 코드

model.fit\_generator() 사용 시

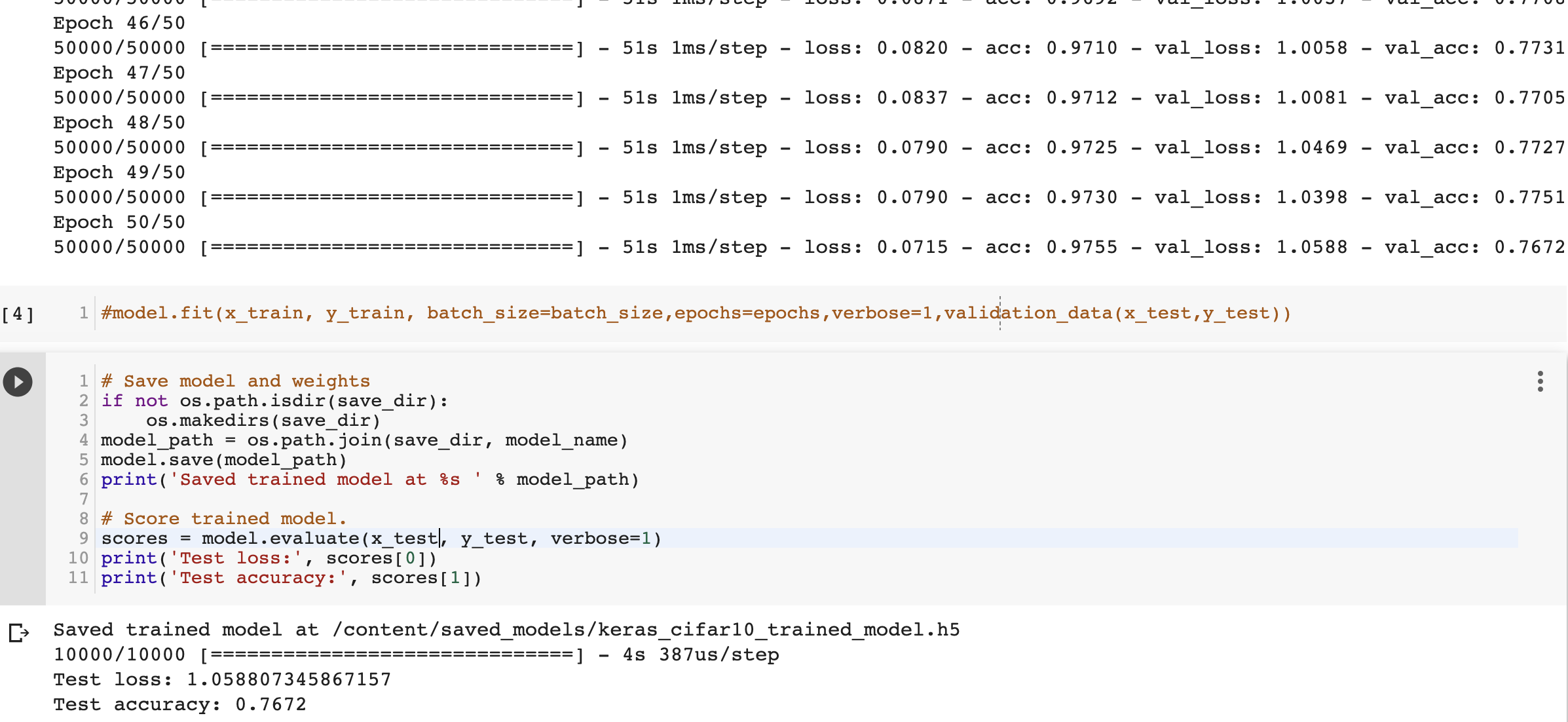


epoch : 1000

Train set accuracy : 0.7811

정확도 : 0.7811

model.fit() 사용 시



epoch : 50

Train set accuracy 0.9755

정확도 : 0.7672

training set 정확도는 97퍼센트에 달하는 것에 비해 정확도가 낮은 것으로 보아, Overfit 된 모델이라고 할 수 있음.

from \_\_future\_\_ import print\_function

import keras

from keras.datasets import cifar10

from keras.utils import Sequence

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers import Conv2D, MaxPooling2D

from IPython.display import SVG

import os

batch\_size = 128

num\_classes = 10

epochs = 10

data\_augmentation = True

num\_predictions = 20

steps\_per\_epoch = 1000

save\_dir = os.path.join(os.getcwd(), 'saved\_models')

model\_name = 'keras\_cifar10\_trained\_model.h5'

# The data, split between train and test sets:

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

print('x\_train shape:', x\_train.shape)

print(x\_train.shape[0], 'train samples')

print(x\_test.shape[0], 'test samples')

# Convert class vectors to binary class matrices.

y\_train = keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = keras.utils.to\_categorical(y\_test, num\_classes)

model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same',

input\_shape=x\_train.shape[1:]))

model.add(Dense(512, activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))

model.add(Dense(512, activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes))

model.add(Activation('softmax'))

optimizer = keras.optimizers.SGD(lr=0.05)

# initiate RMSprop optimizer

opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop

model.compile(loss='categorical\_crossentropy',

optimizer=optimizer,

metrics=['accuracy'])

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

if not data\_augmentation:

print('Not using data augmentation.')

model.fit(x\_train, y\_train,

batch\_size=batch\_size,

epochs=epochs,

validation\_data=(x\_test, y\_test),

shuffle=True)

else:

print('Using real-time data augmentation.')

# This will do preprocessing and realtime data augmentation:

datagen = ImageDataGenerator(

featurewise\_center=False, # set input mean to 0 over the dataset

samplewise\_center=False, # set each sample mean to 0

featurewise\_std\_normalization=False, # divide inputs by std of the dataset

samplewise\_std\_normalization=False, # divide each input by its std

zca\_whitening=False, # apply ZCA whitening

zca\_epsilon=1e-06, # epsilon for ZCA whitening

rotation\_range=0, # randomly rotate images in the range (degrees, 0 to 180)

# randomly shift images horizontally (fraction of total width)

width\_shift\_range=0.1,

# randomly shift images vertically (fraction of total height)

height\_shift\_range=0.1,

shear\_range=0., # set range for random shear

zoom\_range=0., # set range for random zoom

channel\_shift\_range=0., # set range for random channel shifts

# set mode for filling points outside the input boundaries

fill\_mode='nearest',

cval=0., # value used for fill\_mode = "constant"

horizontal\_flip=True, # randomly flip images

vertical\_flip=True, # randomly flip images

# set rescaling factor (applied before any other transformation)

rescale=True,

# set function that will be applied on each input

preprocessing\_function=None,

# image data format, either "channels\_first" or "channels\_last"

data\_format=None,

# fraction of images reserved for validation (strictly between 0 and 1)

validation\_split=0.1)

# Compute quantities required for feature-wise normalization

# (std, mean, and principal components if ZCA whitening is applied).

datagen.fit(x\_train)

# Fit the model on the batches generated by datagen.flow().

model.fit(x\_train, y\_train, batch\_size = batch\_size, epochs = epochs, verbose=1, validation\_data = (x\_test, y\_test))

"""model.fit\_generator(datagen.flow(x\_train, y\_train,

batch\_size=batch\_size),

steps\_per\_epoch=steps\_per\_epoch,

epochs=epochs,

validation\_data=(x\_test, y\_test),

workers=4)"""