목 차

1. 프로그램 개요 3

2. 프로그램 설명 3

3. 모듈 정의 3

4. 소스코드 4

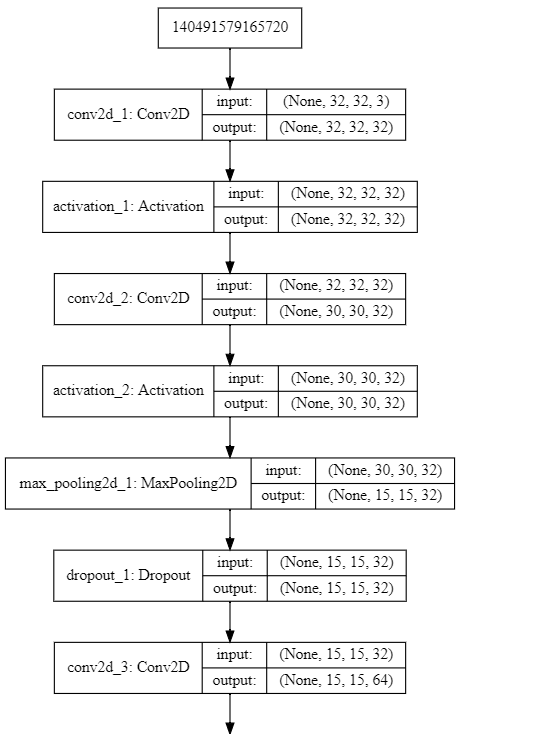
# 프로그램 개요

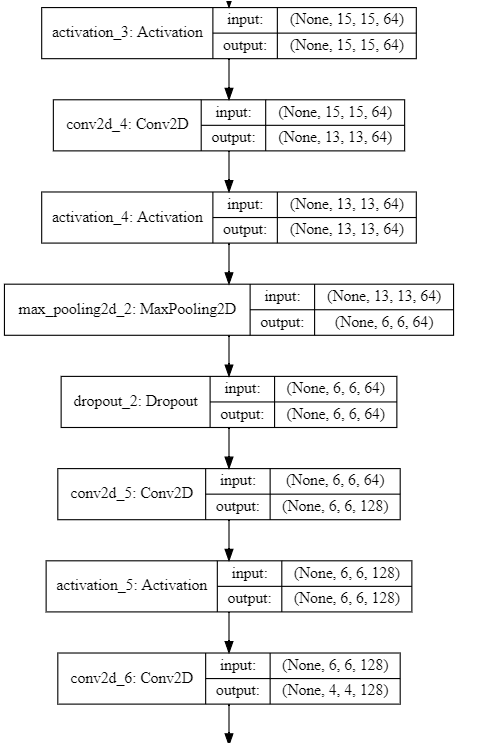
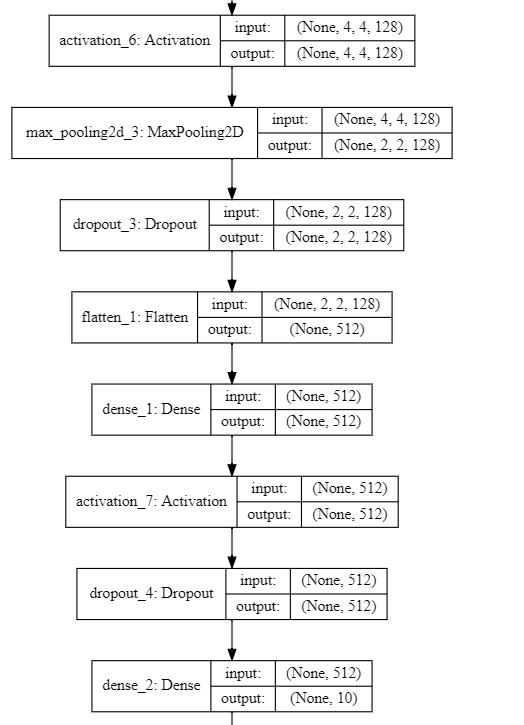
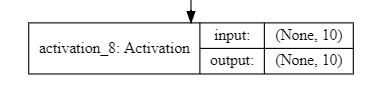
본 프로젝트는 Keras를 이용하여 machine learning을 이해하는 데 목적이 있다. Colab 환경에서 CIFAR-10 데이터 셋을 활용하여 이미지를 10개의 클래스로 분리하는 machine learning을 구현해본다. Convolution Neural Network 에 대한 이해를 바탕으로 Keras에서 제공하는 샘플 코드를 이해하고, 이를 수정하여 정확도 75% 이상을 갖는 프로그램을 구현하는 것이 본 프로젝트의 목표이다.

# 프로그램 설명

CIFAR-10 데이터 셋은 32\*32의 60000개 image로 구성되어 있으며, 10개의 클래스가 각기 6000개의 이미지를 갖는다. 50000개는 training image이며 10000개는 test image이다. CNN은 합성곱 신경망으로, 이미지를 주로 분석하는 데 사용된다. 이미지를 작은 부분으로 나누어 특징을 인식하고 이것을 결합하여 전체를 인식하는 알고리즘을 이용한다. Input레이어, Convolution레이어, Pooling레이어, Output레이어로 구성된다.

cifar10.load\_data() 를 통해 실제 모델을 학습시키는 train 데이터와 성능 평가 시 이용하는 test 데이터를 가져온다.

본 프로그램에서 생성한 모델은 아래와 같다.



1. 학습에 관한 설정값들

Epoch: 학습 횟수. 50000개의 이미지를 모두 한번씩 학습하면 epoch 1회가 된다. 50으로 설정하였다.

Batch\_size: 한 번의 epoch에서 1회에 학습하는 이미지의 개수이다. 한번의 학습에 몇 개의 train 데이터를 사용할 것인지 결정하게 된다. Batch\_size가 작으면 그만큼 1회의 epoch에서 여러 번 학습하게 된다. 256으로 설정하였다.

Learning\_rate: 학습속도. 0과 1 사이의 값으로 설정된다. 가중치를 업데이트 할 때 사용되는데 학습률이 너무 작으면 가중치의 변화가 거의 없어 제대로 갱신되지 않을 수 있다. 0.001로 설정하였다.

1. 모델에 대한 설명

* 모든 convolution레이어의 activation function은 sigmoid가 아닌 relu함수를 사용하였다.
* 1번 convolution 레이어: 3\*3크기의 필터 32개

model.add(Conv2D(32, (3, 3), padding='same',input\_shape=x\_train.shape[1:]))

* Convolution Layer는 입력 데이터로부터 특징을 추출하는 역할을 한다.
* X\_train.shape가 (5000,32,32,3)이므로 이미지 하나의 크기가 32\*32이고, 3은 컬러이미지라는 것을 말한다. 필터를 이동시켜가면서 이미지를 검사한다.
* Padding은 입력 데이터 주변을 특정 값으로 채워 출력 크기를 조정할 목적으로 사용된다. Padding을 same으로 설정하여 입력값과 출력값의 크기가 동일하도록 했다.

model.add(Activation('relu'))

* 2번 convolution 레이어: 3\*3크기의 필터 32개

model.add(Conv2D(32, (3, 3)))

model.add(Activation('relu'))

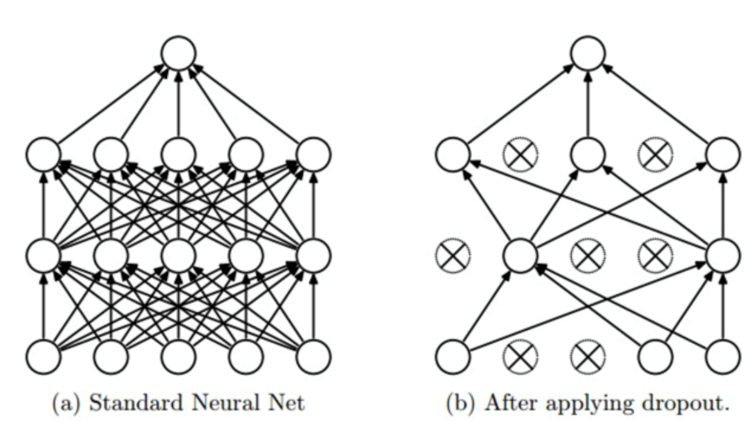
* MaxPooling

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

* 검사한 데이터를 중요한 특성으로 줄이는 maxpooling을 실시한다.
* Dropout

model.add(Dropout(0.25))

* Over fitting을 막기 위한 방법으로, 뉴럴 네트워크가 학습 중일 때 랜덤하게 뉴런을 꺼서 학습을 방해함으로써 학습이 train 데이터에만 잘 적용되도록 치우치는 현상을 막아준다.



* 3번 convolution 레이어: 3\*3크기의 필터 64개

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

* Convolution Layer. 필터 개수를 64개로 설정하여 이미지를 검사한다.

model.add(Activation('relu'))

* 4번 convolution 레이어: 3\*3크기의 필터 64개

model.add(Conv2D(64, (3, 3)))

model.add(Activation('relu'))

* MaxPooling

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

* 다시 max pooling한다.
* Dropout

model.add(Dropout(0.25))

* 5번 convolution 레이어: 3\*3크기의 필터 128개

model.add(Conv2D(128,(3,3), padding='same'))

* Convolution Layer. 필터 개수를 128개로 설정하여 이미지를 검사한다.

model.add(Activation('relu'))

* 6번 convolution 레이어: 3\*3크기의 필터 128개

model.add(Conv2D(128,(3,3)))

model.add(Activation('relu'))

* MaxPooling

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

* 다시 수집된 데이터를 중점적인 정보로 압축하기 위해 Max Pooling을 실행한다.
* DropOut

model.add(Dropout(0,25))

* 임의로 0.25퍼센트의 노드를 꺼준다.
* 일차원의 완전연결층(Fully connected layer)으로 변환

model.add(Flatten())

* Fully connected layer로 변환된다.

model.add(Dense(512))

* Fully connected layer의 개수를 512개로 줄인다.

model.add(Activation('relu'))

* Dropout

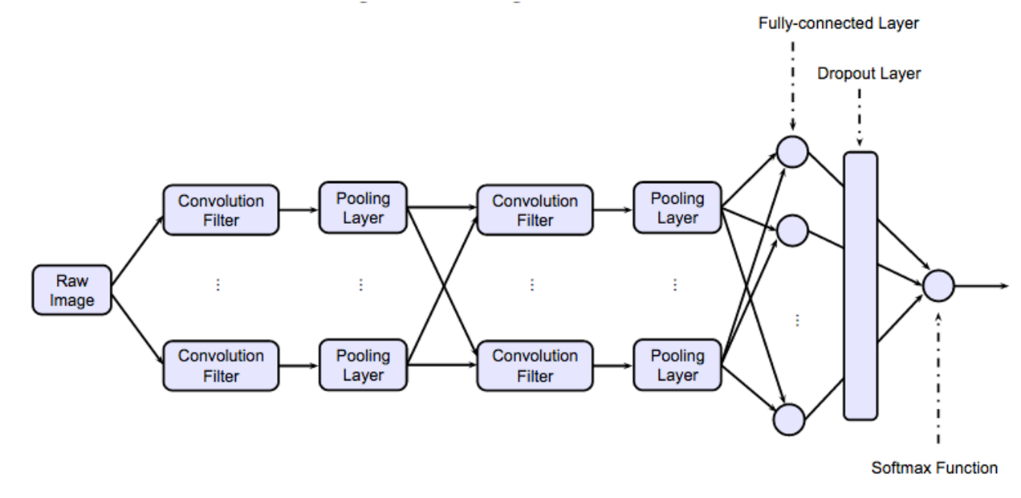
model.add(Dropout(0.5))

model.add(Dense(num\_classes))

* 마지막으로 Fully connected layer의 개수를 10개로 줄여 클래스 10개 중 하나로 구분되도록 한다.

model.add(Activation('softmax'))

* 출력레이어에서는 activation function으로 softmax 함수를 사용하여 일종의 확률값으로 만들어준다. 출력층의 값들을 항상 양수 값을 가지고 모든 값을 더했을 때 1이 되도록 한다.



1. 모델 컴파일

opt = keras.optimizers.rmsprop(lr=learning\_rate, decay=1e-6)

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

optimizer=opt,

metrics=['accuracy'])

10개의 클라스로 나눠지므로 loss함수는 categorical\_crossentropy를 사용한다.

1. 모델 학습

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

batch\_size=batch\_size,

epochs=epochs,

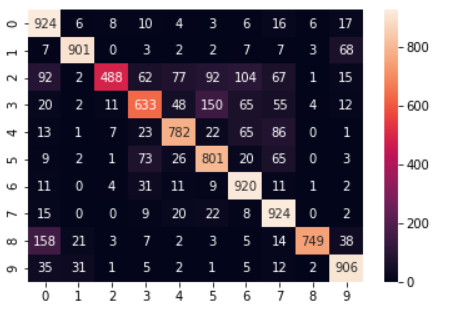
validation\_data=(x\_test, y\_test),

shuffle=True)

epoch은 앞서 언급했듯이 50, batch\_size는256으로 설정하였다.

1. 결과

위의 모델을 이용하여 80%의 정확도를 얻는 학습을 진행할 수 있었다. 아래는 confusion matrix를 heat map으로 보여주는 결과이다.



# 모듈 정의(정확도 향상을 위한 코드개선)

아래의 방법을 통해 80%의 정확도를 달성하는 결과를 도출할 수 있었다.

1. Batch\_size변경

Batch\_size는 한 번에 검사하는 이미지의 개수이다. 테스트 이미지가 50000개이므로, batch size가 256이라면 50000/256=195.xx로, 195번의 트레이닝이 필요하다. Batch\_size가 크면 더 많은 개수의 이미지를 병렬적으로 학습하여 트레이닝 스텝이 적어지기 때문에 실행시간이 줄어들지만, 지나치게 큰 경우 성능이 저하되거나 GPU의 용량을 초과할 수 있다. 따라서 적당한 값인 256으로 설정하였다.

1. Epoch 변경

Epoch은 학습하는 횟수이다. Epoch을 증가시킬수록 정확도가 높아질 것이지만 100번은 시간이 매우 길게 소요되어 80%의 정확도를 달성하는 epoch 50으로 설정하였다.

1. Convolution 레이어 추가

3\*3 크기의 필터 128개로 학습하는 convolution layer을 추가하였다.

레이어를 깊게 쌓게 되면 얕은 뉴럴 네트워크보다 더 강력한 학습모델을 구현할 수 있다.

초반의 convolution layer에서 line, edge와 같은 low-level의 부분적인 특징을 추출했다면 더 높은 단계의 레이어는 점차적으로 추상적인 물체와 같은 특징을 추출하게 되어 더 정확한 값을 얻을 수 있게 된다.

새로 추가한 Convolution 레이어 뒤에 Maxpooling레이어도 또한 추가하였다.

#5번 convolution 레이어: 필터 128개

model.add(Conv2D(128,(3,3), padding='same'))

model.add(Activation('relu'))

#6번 convolution 레이어: 필터 128개

model.add(Conv2D(128,(3,3)))

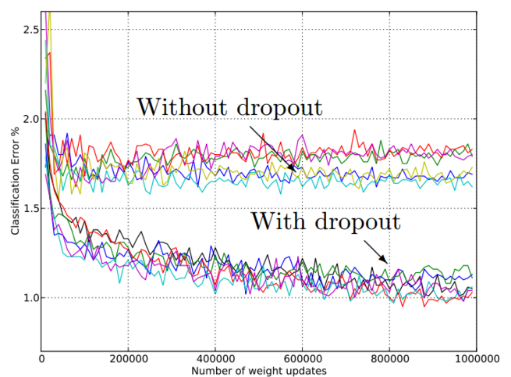
model.add(Activation('relu'))

#MaxPooling

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

model.add(Dropout(0,25))

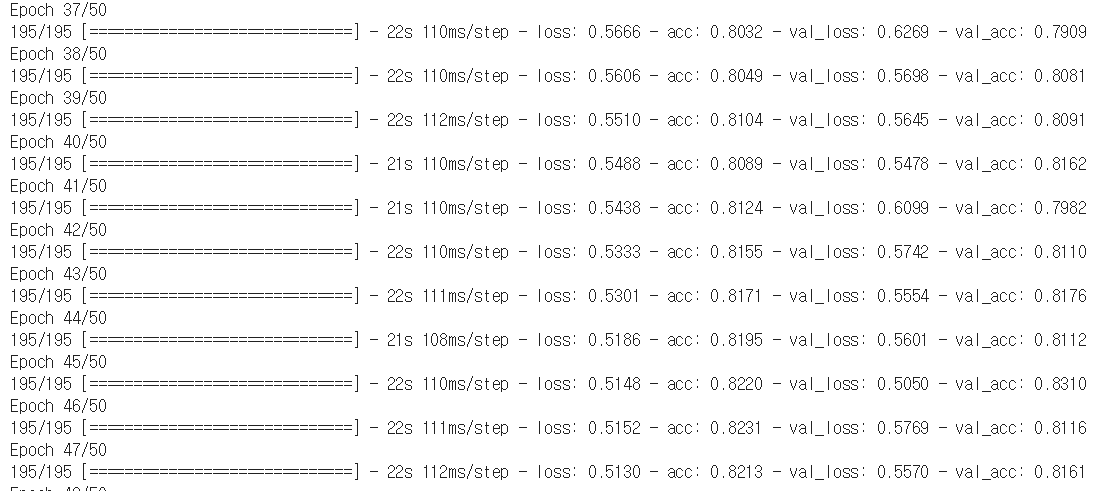
1. Drop out 레이어 추가

Drop out레이어를 추가하면 학습이 train데이터에만 치우치는 것을 방지해주어 test데이터나 외부 데이터에 적용했을 때 더 정확한 값을 얻을 수 있다. 일반적으로 CNN에서는 drop out레이어를 fully connected network 뒤에 놓는데, 상황에 따라 max pooling 계층 뒤에 놓기도 한다. 본 프로젝트에서는 새로 추가한 필터 128개의 convolution layer 뒤의 max pooling 레이어 뒤에 하나의 drop out 레이어를 추가하였다. 아래 표는 dropout레이어를 추가했을 때 에러 발률이 적어짐을 보여준다.

1. 이외에도 learning rate변경 시 더 높은 정확도를 얻을 수 있음

본 프로젝트에 적용하지 않았지만, 이 프로그램의 learning rate가 0.001로 설정되어 있는 것을 0.0005로 더 적게 하면 더 높은 정확도를 얻을 수 있다. Learning rate는 weight를 얼만큼 수정할 것인지를 결정하는 값인데, learning rate가 지나치게 크면 weight가 크게 수정되어 엉뚱한 값이 도출될 수 있고, 지나치게 작으면 weight가 거의 갱신되지 않아 학습이 올바르게 이루어지지 않을 수 있다. 따라서 적당한 값을 찾는 것이 중요하다. 본 프로젝트에서 learning rate를 0.0005로 수정하였을 때, epoch 37번째에서 이미 정확도 80%를 달성하였고, epoch 47번째에서 정확도 82%를 달성함을 확인할 수 있었는데 시간 관계상 기존의 learning rate 0.001로 프로그램을 마무리하였다.

---learning rate 0.0005적용시 예상 결과



# 코드

# -\*- coding: utf-8 -\*-

"""CIFAR10.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1gZ-r0QUK3c8YM5hGD-Z7mmhYNWjCSyKX

\*\*Import Modules\*\*

"""

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

import keras

from keras.datasets import cifar10

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

# %matplotlib inline

from keras.utils.vis\_utils import model\_to\_dot

from sklearn.metrics import confusion\_matrix

import pandas as pd

import seaborn as sns

batch\_size = 256 #한번에 학습하는 이미지의 개수

num\_classes = 10 #cifar10의 클래스의 개수는 10개

epochs = 50 #학습의 횟수

learning\_rate=0.001

data\_augmentation = True

num\_predictions = 20

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

"""\*\*Load Data set\*\*"""

(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')

"""\*\*One Hot Encoding [1,0,0,0,0,0,0,0,0,0]의 형태로 만들어준다\*\*"""

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

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

"""\*\*Function to print images and answers\*\*"""

import matplotlib.pyplot as plt

def plot\_images(x, y\_true, y\_pred=None, size=(5,5)):

assert len(x) == len(y\_true) == size[0] \* size[1]

fig, axes = plt.subplots(size[0], size[1])

fig.subplots\_adjust(hspace=1, wspace=4)

for i, ax in enumerate(axes.flat):

if x[i].shape[-1] ==1:

ax.imshow(x[i].reshape(x[i].shape[0], x[i].shape[1]))

else:

ax.imshow(x[i])

if y\_pred is None:

xlabel = "True: {0}".format(y\_true[i].argmax())

else:

xlabel = "True: {0}, Pred: {1}".format(y\_true[i].argmax(),

y\_pred[i].argmax())

ax.set\_xlabel(xlabel)

ax.set\_xticks([])

ax.set\_yticks([])

plt.show()

plot\_images(x\_train[:25], y\_train[:25])

"""\*\*Create model\*\*"""

model = Sequential()

#convolution layer.

#input\_shape = 32\*32, color image

#1번 convolution 레이어: 필터 32개

model.add(Conv2D(32, (3, 3), padding='same',input\_shape=x\_train.shape[1:]))

#padding='same'은 출력크기와 입력크기를 같게 하는 것임

model.add(Activation('relu'))

#2번 convolution 레이어: 필터 32개

model.add(Conv2D(32, (3, 3)))

model.add(Activation('relu'))

#MaxPooling

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

#과적합 방지

model.add(Dropout(0.25))

#3번 convolution 레이어: 필터 64개

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

model.add(Activation('relu'))

#4번 convolution 레이어: 필터 64개

model.add(Conv2D(64, (3, 3)))

model.add(Activation('relu'))

#MaxPooling

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

model.add(Dropout(0.25))

#5번 convolution 레이어: 필터 128개

model.add(Conv2D(128,(3,3), padding='same'))

model.add(Activation('relu'))

#6번 convolution 레이어: 필터 128개

model.add(Conv2D(128,(3,3)))

model.add(Activation('relu'))

#MaxPooling

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

model.add(Dropout(0,25))

#일차원의 완전연결층으로 변환

model.add(Flatten())

#512개로 줄임

model.add(Dense(512))

model.add(Activation('relu'))

model.add(Dropout(0.5))

#마지막으로 10개의 클래스로 줄임. 출력층

model.add(Dense(num\_classes))

model.add(Activation('softmax'))

model.summary()

"""\*\*Optimizer: learing\_rate는 학습속도, decay는 학습 속도를 감소시키는 옵션\*\*"""

opt = keras.optimizers.rmsprop(lr=learning\_rate, decay=1e-6)

"""\*\*Compile model\*\*"""

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

optimizer=opt,

metrics=['accuracy'])

"""\*\*데이터 정규화\*\*"""

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

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

x\_train /= 255

x\_test /= 255

"""\*\*학습시킴\*\*"""

if not data\_augmentation:

print('Not using data augmentation.')

#batch\_size 256, epoch=50으로 설정

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=False, # randomly flip images

# set rescaling factor (applied before any other transformation)

rescale=None,

# 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=0.0)

# 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\_generator(datagen.flow(x\_train, y\_train,

batch\_size=batch\_size),

epochs=epochs,

validation\_data=(x\_test, y\_test),workers=4, steps\_per\_epoch=x\_train.shape[0]//batch\_size)

"""\*\*테스트 셋으로 평가\*\*"""

scores = model.evaluate(x\_test, y\_test, verbose=1)

print('Test loss:', scores[0])

print('Test accuracy:', scores[1])

y\_pred = model.predict(x\_test)

plot\_images(x=x\_test[:25], y\_true=y\_test[:25], y\_pred = y\_pred[:25])

y\_result = confusion\_matrix(y\_test.argmax(axis=1), y\_pred.argmax(axis=1))

sns.heatmap(pd.DataFrame(y\_result, range(10), range(10)), annot=True, fmt='g')

from IPython.display import SVG

SVG(model\_to\_dot(model, show\_shapes=True).create(prog='dot', format='svg'))