**과목명: 시스템프로그래밍**

**2 분반**

**<<Project #4>>**

**서강대학교 [학부명] 컴퓨터공학과**

**[학번] 20171640**

**[이름] 박수진**

목 차

1. **프로그램 개요**
2. **프로그램 설명**
   1. 프로그램 실행 방법
   2. 모델 구성
   3. 모델 향상 방법
   4. confusion matrix
   5. loss function
   6. rank
3. **소스 코드**

# 프로그램 개요

CIFAR-10 data set은 60000개의 32\*32 픽셀 컬러 이미지로, 클래스당 6000개씩 총 10개의 클래스로 이루어져 있다. 개발 환경은 Google Colab에서 Keras를 이용한 개발로, MNIST data set에 최적화된 코드를 수정하여 CIFAR-10 data set에서 80% 이상의 정확도를 얻어낼 수 있는 CNN 프로그램을 작성한다.

# 프로그램 설명

## 프로그램 실행 방법

1. Google Colab에서 [파일]>[노트 열기](단축키 Ctrl+O)로 노트북 파일(20171640.ipynb)을 불러온다.

2-1. 프로그램 전체를 실행할 시에는 [런타임]>[모두 실행] (단축키 Ctrl+F9)로 실행한 후 결과를 확인할 수 있다.

2-2. 현재 선택 중인 셀만 실행하려면 [런타임]>[초점이 맞춰진 셀 실행](단축키 Ctrl+Enter)로 실행할 수 있다.

3. 현재 런타임을 초기화하려면 [런타임]>[런타임 초기화]로 초기화할 수 있다.

## 모델 구성

**#Adding layers to the model**

model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu',

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

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(BatchNormalization())

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

model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(Conv2D(256, (3, 3), activation='relu'))

model.add(BatchNormalization())

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

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

**\* Background**

- 3차원 a\*b\*c input layer에 n개의 x\*y\*z size convolution filter를 적용하는 경우 3차원 output layer의 size는 ((a-x)/stride + 1) \* ((a-x)/stride + 1) \* n이 된다.

- 3차원 a\*b\*c input layer에 x\*y Max pooling을 적용하면 3차원 output layer의 size는 (a/x) \* (b/y) \* c가 된다.

- 3차원 a\*b\*c input layer를 flatten하면 output layer는 1차원이 되며 그 size는 a\*b\*c이다.

- a size를 갖는 1차원의 input layer에 dense 값 x를 적용하면 1차원 output layer의 size는 x가 된다.

- dropout은 training 과정에서 뉴런의 연결을 임의로 삭제하여 overfitting을 방지하는 기법이다.

파라미터는 dropout하는 ratio를 나타내며 input layer와 ouput layer의 dimension 및 size 차이는 없다.

- batchnormalization은 input을 안정화하여 레이어가 더 안정적으로 학습할 수 있도록 한다. input layer와 output layer의 diemnsion 및 size 차이가 없다.

**\* layer 정보 및 순서**

1. 제일 처음 아무 과정을 거치지 않은 본래의 input layer는 3차원 dimension의 32\*32\*3 size를 갖는다.

2. 32개의 3\*3 size kernel을 갖는 convolution filter를 적용한다. (activation funciton : ReLU)

input layer의 dimension은 32\*32\*3이므로 (32-3)/1 + 1 = 30, 즉 output layer는 30\*30의 size를 갖는다. filter의 개수가 32개이므로 depth는 32가 되어 최종적으로 output layer는 30\*30\*32의 dimension을 갖는다.

3. 64개의 3\*3 size kernel을 갖는 convolution filter를 적용한다. (activation funciton : ReLU)

input layer의 dimension은 30\*30\*32이므로 (30-3)/1 + 1 = 28, 즉 output layer는 28\*28의 size를 갖는다. filter의 개수가 64개이므로 depth는 64가 되어 최종적으로 output layer는 28\*28\*64의 dimension을 갖는다.

4. batchnormalization을 적용한다. input output의 dimension 및 size 차이는 없다.

5. 2\*2로 Maxpooling을 적용한다. input layer는 28\*28\*64이므로 2\*2 max pooling을 적용하면 28/2 = 14가 되어 output layer는 14\*14\*64가 된다.

6. 0.25 ratio로 dropout한다. input과 ouput의 dimension 및 size 차이는 없다.

7. 128개의 3\*3 size kernel을 갖는 convolution filter를 적용한다. (activation funciton : ReLU)

input layer의 dimension은 14\*14\*64이므로 (14-3)/1 + 1 = 12, 즉 output layer는 12\*12의 size를 갖는다. filter의 개수가 128개이므로 depth는 128이 되어 최종적으로 output layer는 12\*12\*128의 dimension을 갖는다.

8. 256개의 3\*3 size kernel을 갖는 convolution filter를 적용한다. (activation funciton : ReLU)

input layer의 dimension은 12\*12\*128이므로 (12-3)/1 + 1 = 10, 즉 output layer는 10\*10의 size를 갖는다. filter의 개수가 256개이므로 depth는 256이 되어 최종적으로 output layer는 10\*10\*256의 dimension을 갖는다.  
9. batchnormalization을 적용한다. input output의 dimension 및 size 차이는 없다.

10. 2\*2로 Maxpooling을 적용한다. input layer는 10\*10\*256이므로 2\*2 max pooling을 적용하면 10/2 = 5가 되어 output layer는 5\*5\*256가 된다.

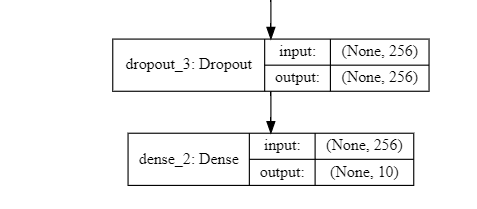
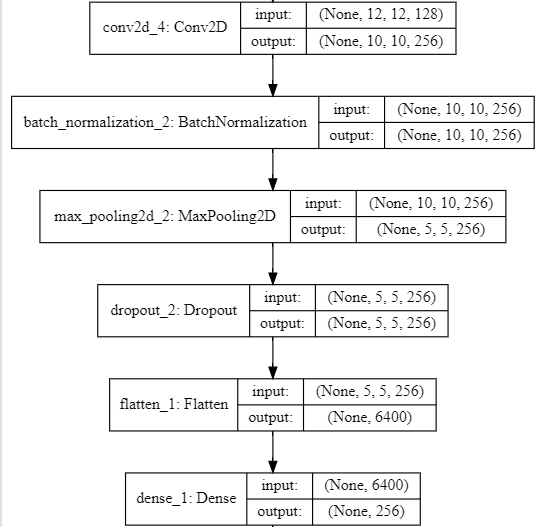
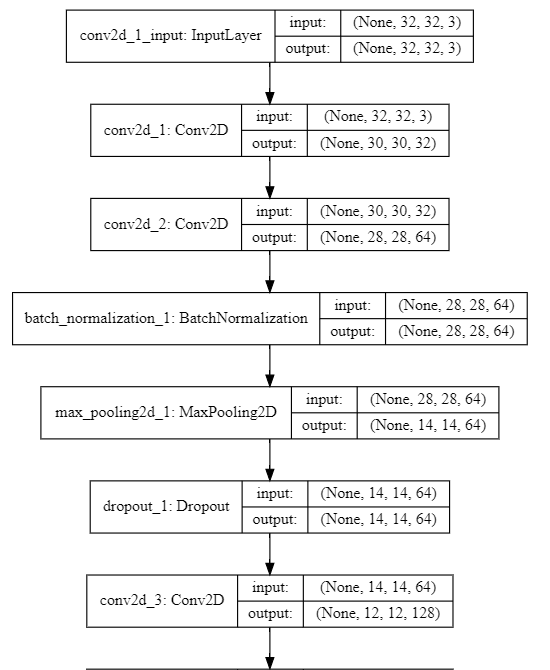
11. 0.25 ratio로 dropout한다. input과 ouput의 dimension 및 size 차이는 없다.

12. 3차원 dimension input layer를 1차원으로 flatten한다. input layer가 5\*5\*256이므로 flatten한 결과는 5\*5\*256 = 6400이 된다. 즉 output layer는 1차원 6400 size를 갖는다

13. activation fuction으로 ReLU를 사용하는 FC layer를 적용한다. input layer size 6400에서 dense 값 256 적용하여 output layer size 256의 결과를 얻는다.

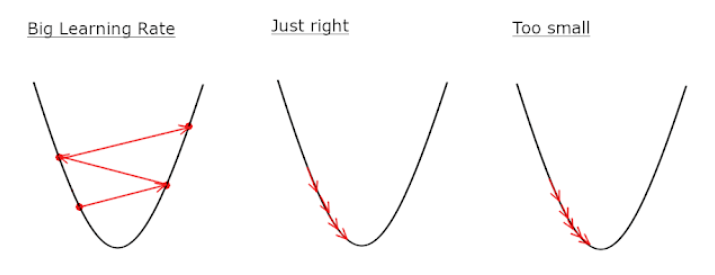
14. 0.5 ratio로 dropout한다. input과 ouput의 dimension 및 size 차이는 없다.

15. activation fuction으로 softmax를 사용하는 FC layer를 적용한다. input layer size 256에서 dense 값 num\_classes(클래스 개수 10개) 적용하여 output layer size 10의 결과를 얻는다.



## 모델 향상 방법

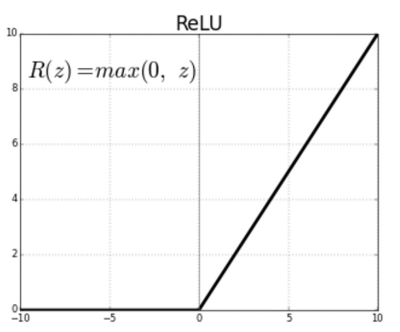
1. learning rate를 0.05로 올렸다. learning rate가 너무 크면 이동 거리가 커서 데이터가 무질서하게 이탈하여 최저점에 수렴하지 못하며, 너무 작으면 조금씩 이동하기 때문에 학습 시간이 오래 걸려서 최저점에 도달하지 못한다. 따라서 적절한 learning rate를 찾아주는 것이 중요하다.



2. 더 많은 layer층을 쌓았다. CNN을 통해 learning을 하는데, depth가 256을 넘어가면 accuracy가 떨어지는 것을 확인하고 Convolution의 filter 개수를 256까지만 늘렸다. Pooling은 최댓값을 뽑아내서 size를 줄이는 Max pooling을 사용하였고, 마지막에는 classification을 위해 FC layer를 추가해주었다. 쌓은 layer층에 대한 자세한 설명은 2-2에서 확인할 수 있다.

3. batch normalization을 적용하였다. batch normalization을 사용하면 gradient vanishing, gradient exploding을 줄일 수 있다. 각 layer에 들어가는 input을 normalize 시켜서 layer의 학습을 가속화한다. 이 때 mini-batch의 평균과 분산을 이용하여 normalize한다. dropout과 비슷한 효과를 갖고 있지만 학습 속도가 느려지는 dropout의 단점에서 벗어날 수 있다.

4. ReLU가 CNN에 적합한 activation function이기 때문에 ReLU를 사용하였다. ReLU는 0 미만의 값은 0으로 처리한다.

(ReLU activation function)

최종 output layer에는 softmax activation function을 사용하였다.

5. Adam optimizer를 사용하였을 때 accuracy가 감소하는 것을 확인하였다. SGD optimizer로도 80% 이상의 accuracy를 얻을 수 있었기 때문에 SGD optimizer를 사용하였다. SGD optimizer는 일부 데이터만 계산하면서 스텝을 빠르게 전진하기 때문에 빠르게 결과를 얻을 수 있다.

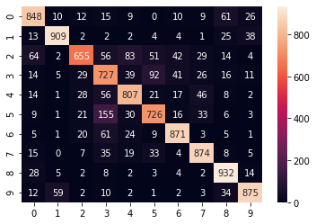
6. overfitting을 방지하기 위해 뉴런의 연결을 임의로 삭제하는 dropout을 사용하였다.

위와 같은 방법으로 학습한 모델의 evaluation은 다음과 같다.

Test loss: 0.7254315194368363

Test accuracy: 0.8223999738693237

## confusion matrix



## loss function

model.compile(loss=keras.losses.categorical\_crossentropy,

              optimizer=optimizer,

              metrics=['accuracy'])

loss function을 통해 실제 데이터와 학습을 통해 얻은 데이터의 추정치가 얼마나 차이나는 지에 대해 evaluation할 수 있다. 이 모델에서는 categorical\_crossentropy를 loss function으로 사용하였다. cross entropy function은 다음과 같다.



one-hot encoding에 사용되며 주로 분류 문제에 사용된다. yi는 실제 분포를 나타내며 log 안의 내용은 학습을 통해 얻은 추정치이다. 는 softmax 함수로 주어진 벡터의 값들을 [0,1]에 bound 시킨다. one-hot encoding이란 하나의 값만 1(True)이고 나머지는 모두 0(False)인 인코딩이다. 예를 들어 MNIST dataset의 경우 3을 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]의 형식으로 저장하는 것이다. 학습에 사용되는 CIFAR-10 data set의 경우 10개의 클래스로 이루어져 있으므로 다중 분류 손실 함수인 categorical\_crossentropy를 사용한다.

## rank

if len(x\_train.shape) < 4:

  x\_train = x\_train.reshape(x\_train.shape[0], x\_train.shape[1], x\_train.shape[2], 1)

  x\_test = x\_test.reshape(x\_test.shape[0], x\_train.shape[1], x\_train.shape[2], 1)

training 과정에서 input을 3 dimension(width\*height\*depth)으로 다루기 위해 image data set의 rank를 4로 설정하여 전처리한다. 이 때 데이터는 (학습 데이터 수, width(pixel 수), height(pixel 수), channel 수)의 정보를 포함한다.

CIFAR-10 data set의 경우 color image로써 R, G, B의 3 channel을 갖는다. reshape 전처리 과정을 거치지 않은 상태로 CIFAR-10의 shape를 출력해보면 (50000, 32, 32, 3)으로 출력되는데, 이를 통해 CIFAR-10은 채널 값이 1보다 크므로 이미지를 로드할 때부터 채널 개수가 포함된 rank 4의 데이터로 로드되어 reshape 할 필요가 없다는 것을 알 수 있다. 실제로 CIFAR-10은 len(x\_train.shape) == 4 이기 때문에 reshape하는 if문 안으로 들어가지 않는다.

그러나 MNIST data set의 경우 전처리 하지 않은 상태로 shape을 출력하면 (60000, 28, 28)의 rank 3 데이터로 출력되는데, 이는 MNIST data set이 grayscale(흑백) 1 channel image이기 때문에 channel 값이 생략된 상태로 data가 load된 것이다. 즉, MNIST data set은 len(x\_trainshape) == 3이므로 if문 안에서 rank 4로 reshape가 필요하다. MNIST의 channel 개수인 1을 마지막에 넣어서 reshape 해준다.

만약 이 전처리 과정을 거치지 않으면, channel 값이 누락되었을 경우 input의 depth를 알 수 없기 때문에 training 과정에서 문제가 발생한다. 그러므로 rank 4로 reshape가 반드시 필요하다.

# 소스 코드

**#Import modules**

import keras

from keras.datasets import cifar10

from keras.models import Sequential

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

from keras.layers import Conv2D, MaxPooling2D

from keras.layers.normalization import BatchNormalization

from keras import backend as K

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

from IPython.display import SVG

%matplotlib inline

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

import pandas as pd

import seaborn as sns

**#Tuning parameters**

epochs = 50

learning\_rate=0.05

batch\_size = 128

num\_classes = 10

**#Plot Image**

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=0.5, wspace=0.1)

    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()

**#Load dataset**

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

**#Reshaping the data**

if len(x\_train.shape) < 4:

  x\_train = x\_train.reshape(x\_train.shape[0], x\_train.shape[1], x\_train.shape[2], 1)

  x\_test = x\_test.reshape(x\_test.shape[0], x\_train.shape[1], x\_train.shape[2], 1)

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train /= 255

x\_test /= 255

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

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

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

**#Applying One hot encoding for the data**

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

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

**#Show data**

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

**#Creating the DNN model**

model = Sequential()

**#Adding layers to the model**

model.add(Conv2D(32, kernel\_size=(3, 3), activation='relu',

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

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(BatchNormalization())

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

model.add(Dropout(0.25))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(Conv2D(256, (3, 3), activation='relu'))

model.add(BatchNormalization())

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

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

**#Visualization the model**

SVG(model\_to\_dot(model, show\_shapes=True, dpi=70).create(prog='dot', format='svg'))

**#Optimizer**

optimizer = keras.optimizers.SGD(lr=learning\_rate)

**#Compiling the model**

model.compile(loss=keras.losses.categorical\_crossentropy,

              optimizer=optimizer,

              metrics=['accuracy'])

**#Training the model**

model.fit(x\_train, y\_train,

          batch\_size=batch\_size,

          epochs=epochs,

          verbose=1,

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

**#Evaluating the model**

score = model.evaluate(x\_test, y\_test, verbose=0)

print('Test loss:', score[0])

print('Test accuracy:', score[1])

**#Prediction the model**

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')