**과목명: 시스템프로그래밍**

**분반: CSE4100 - 01**

**Project #4**

담당교수: 서강대학교 컴퓨터공학과 박운상

서강대학교

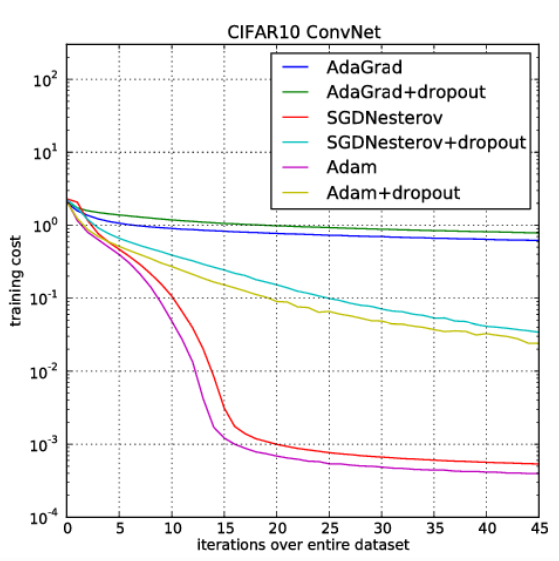
학번: 20170001

이름: 강서현

**목차**

1. **프로젝트 목표**
2. **모델 구성**
   1. 시도한 방법
   2. 완성 모델
3. **코드 구현**
   1. 모듈 import(실행 방법)
   2. 데이터 전처리(Rank 재설정)
   3. 학습 과정(loss function)
   4. 데이터 예측(Confusion matrix)
4. 프로젝트 개요

MNIST 예제를 바탕으로 Keras를 이용하여 이미지 분류를 하는 머신 러닝모델을 개발한다. CIFAR-10 데이터 셋에 대한 정확도를 80% 이상 높이는 것을 목표로 한다.

1. 모델 구현
   1. 시도한 방법
      1. Optimizer 변경

Keras에서 제공하는 Optimizer는 SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, Nadam, Ftrl 이다. 이 중 Adam이 가장 보편적이고 성능이 어느정도 보장된다고 하여 Adam을 적용하였다. 좌측 그림처럼 CIFAR-10를 학습할 때 Adam의 training cost가 AdaGrad나 SGDNesterov보다 적다고 한다. learning rate = 0.001로 Adam을 적용한 결과 test accuracy가 0.66에서 0.726으로 상승하였다.(epoch 10일 때)

출처: Adam: A Method for Stochastic Optimization

* + 1. Learning rate 변경

다음으로는 learning rate를 변경하면서 성능 변화를 확인해 보았다. learning rate가 0.0001일 때 test accuracy가 0.66으로, lr가 0.002에서는 0.712으로 변화했다. learning rate가 0.0001일 때는 optimizer가 없을 때와 성능이 비슷한 것으로 보아, weight 업데이트가 이뤄지지 않는 것 같다. 또한 o.oo2일 때는 0.001일 때보다 성능이 좋지 않았다. 또한 일반적으로 keras에서 Adam을 사용할 때는 0.001의 learning rate를 사용한다고 하여 처음에 설정한 0.001을 그대로 사용하게 되었다.

* + 1. Layer 쌓기

여러 개의 layer를 쌓아보면서 성능을 살펴보았다. 초반에는 아래 코드와 같이 Dense layer직전에 커널 개수가 많은 convolution layer를 더 쌓는 방식으로 해결하려고 하였다. 즉 기존의 코드에 노란색 부분만 추가한 것이다. 이처럼 구현하면 짧은 시간에 accuracy는 상당히 증가하지만(0.726에서 0.76) overfitting 될 가능성이 매우 커진다. 이는 loss 그래프에서 epoch7이 지나자 train loss는 계속 감소하는 반면 test 세트는 loss가 정체되는 부분에서 확인할 수 있었다.

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(Conv2D(128, (3, 3), activation='relu'))

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

model.add(Dropout(0.25))

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

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

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

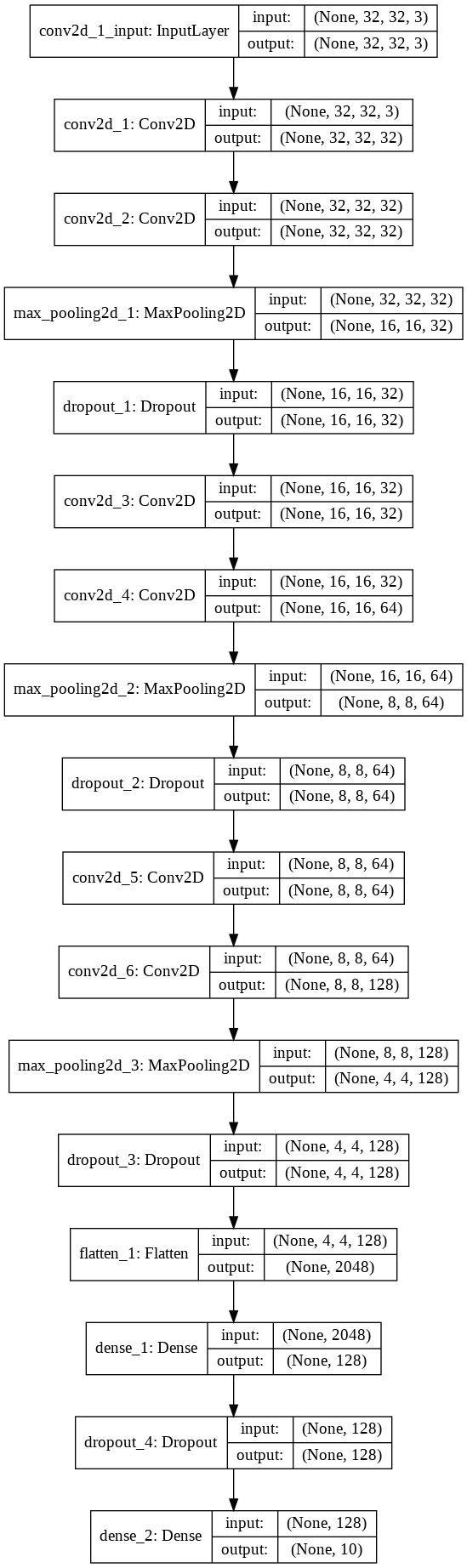
model.add(Dense(num\_classes, activation='softmax'))

따라서 epoch이 길어져도 안정적으로 loss 곡선이 나오도록 다른 방법을 찾아보았고, convolution과 maxpooling, dropout을 함께 추가하는 방식으로 구현해보았다. convolution 2개와 maxpooling 1개, dropout 1개의 layer를 하나의 묶음으로 생각하면, 총 세 묶음으로 구현하고, 마지막에 dense layer를 거치는 것이다. 그 결과 이전 방식보다는 안정적으로, 꾸준히 정확도가 증가하는 것을 볼 수 있었다.

* 1. 완성 모델

앞서 설명한 것처럼 Conv2D 2개와, MaxPooling2D 1개, Dropout 1개로 이루어진 묶음을 3개 넣고, 마지막에 dense layer를 구현하였다. 이때, 기존 코드에 나타나 있던 것처럼 뒤의 layer로 갈수록 커널 크기가 증가하며, activation 함수는 relu로 고정하였다. 한편, 모든 Conv2D에는 padding을 주어서 데이터의 크기가 변하지 않도록 하였다. 이는 layer를 거치며 데이터 크기가 계속 작아지게 되면 충분히 많은 layer를 쌓지 못하기 때문에 추가한 것이다. MaxPooling시에는 pool 사이즈를 (2,2)로 고정하여 최대한 이전 데이터를 유지할 수 있도록 했다. 그리고 Dropout을 많이 추가하여 overfitting 문제가 최소화되도록 하였다.

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

32개의 3x3 커널의 convolution filter가 있는 layer이다. 이 layer의 input은 기존 Cifar 10 데이터의 크기인 (None, 32, 32, 3)이다. padding이 기존 dimension을 유지해주기 때문에 channel 크기만 kernel 개수로 변경된다. 따라서 output은 (None, 32, 32, 32)이다.

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

앞 layer와 동일한 layer이다. 따라서 input과 output 모두 (None, 32, 32, 32)이다.

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

2x2 커널 크기의 max polling layer이다. row, col 각각 반으로 줄어들기 때문에 input은 (None, 32,32,32), output은 (None, 16,16,32)이다.

model.add(Dropout(0.25))

학습 과정 중에 랜덤하게 일부 노드들을 생략하고 학습하는 것이다. 이는 overfitting 문제를 해결하는 데에 도움이 되는 것으로 알려져 있다.

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

앞선 convolution layer와 동일하다. input과 output dimension이 (None, 16,16,32)으로 같다.

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

64개의 3x3 커널의 convolution filter가 있는 layer이다. padding으로 row, col의 크기는 유지되지만 kernel 개수 때문에 channel 사이즈가 변경된다. 따라서 input (None, 16,16,32), output (None, 16,16,64)이다.

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

앞선 MaxPooling layer와 동일하다. row, col 각각 반으로 줄어들기 때문에 input은 (None, 16,16,64), output은 (None, 8, 8, 64)이다.

model.add(Dropout(0.25))

앞선 dropout layer와 동일하다.

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

앞선 convolution layer와 동일하다. input과 output dimension이 (None, 8, 8, 64)로 같다.

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

128개의 3x3 커널의 convolution filter가 있는 layer이다. padding으로 row, col의 크기는 유지되지만 kernel 개수 때문에 channel 사이즈가 변경된다. 따라서 input (None, 8, 8, 64), output (None, 8,8,128)이다.

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

앞선 MaxPooling layer와 동일하다. row, col 각각 반으로 줄어들기 때문에 input은 (None, 8,8,128), output은 (None, 4,4,128)이다.

model.add(Dropout(0.5))

앞선 dropout layer와 동일하다.

model.add(Flatten())

4(3) dimension을 2(1) dimension으로 변경하는 과정이다. 따라서 (None, 4,4,128)가 (None, 2048)로 바뀐다.

model.add(Dense(128, activation='relu'))

output 노드가 128인 fully-connected layer로 연산을 수행한다. 따라서 input (None, 2048)는 output (None, 128)로 바뀐다.

model.add(Dropout(0.5))

앞선 dropout layer와 동일하다.

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

softmax로 마지막 분류 작업을 수행한다. num\_classes가 10이므로 input은 (None, 128), output은 (None, 10)이 된다.

1. 코드 구현
   1. 모듈 import

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 import backend as K

from keras.utils.vis\_utils import model\_to\_dot, plot\_model #loss graph를 그리기 위해 사용

from keras.optimizers import Adam #Adam optimizer 사용

먼저 keras를 사용하기 위해 keras를 import한다.

keras.datasets에서 cifar 10을 가져오고, 모델 레이어를 구현하기 위해 keras.models에서 Sequential을, keras.layers에서 사용할 모듈을 import하였다. 한편 loss 그래프를 그리기 위해 keras.utils.vis\_utils에서 plot\_model을 가져왔으며, Adam optimizer를 사용하기 위해 keras.optimizers에서 Adam을 import하였다.

epoch은 50으로 고정하였고, learning rate는 2.1.2에서 다룬 것처럼 0.001로 하였다.

* 1. 데이터 전처리(Rank 재설정)

if len(x\_train.shape) < 4:  #dimension 맞추기

  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)

데이터를 로드한 다음, 이미지를 담은 배열의 rank가 4인지 확인하고 4가 아니라면 재설정하는 부분이 있다. 이는 convolution 2d 모듈(Conv2D)의 input shape으로  (batch\_shape, rows, cols, channels)가 입력되어야 하기 때문이다. 만일 mnist 데이터와 같이 이미지가 흑백이라면, 한 개의 channel로 표현될 수 있기 때문에 channel에 해당하는 dimension이 없을 수 있다. 이 경우 마지막 dimension에 1을 넣어 reshape을 해주어야 Conv2D모듈을 사용할 수 있다.

x\_train /= 255 #normalization

x\_test /= 255

한편 데이터의 숫자를 0과 1사이의 수로 변경하기 위해 255로 나누어 주었다.(Normalization)

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

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

또한, 출력에 on-hot-encoding을 적용하여, 답에는 1을, 나머지에는 0을 대입하였다.

* 1. 학습 과정(loss function)

모델은 2.2에서 자세히 설명했으므로 생략한다. model = Sequential()로 모델을 생성할 수 있으며, model.summary()로 모델 요약 부분을 출력하였다.

2에서 언급한 것처럼 컴파일시 Adam optimizer를 사용하였으며, learning rate는 0.001로 하였다.

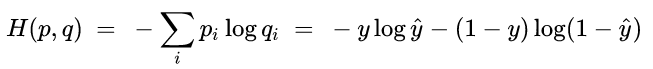
model.compile(loss=keras.losses.categorical\_crossentropy,

              optimizer=optimizer,

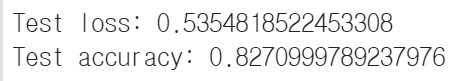
              metrics=['accuracy'])

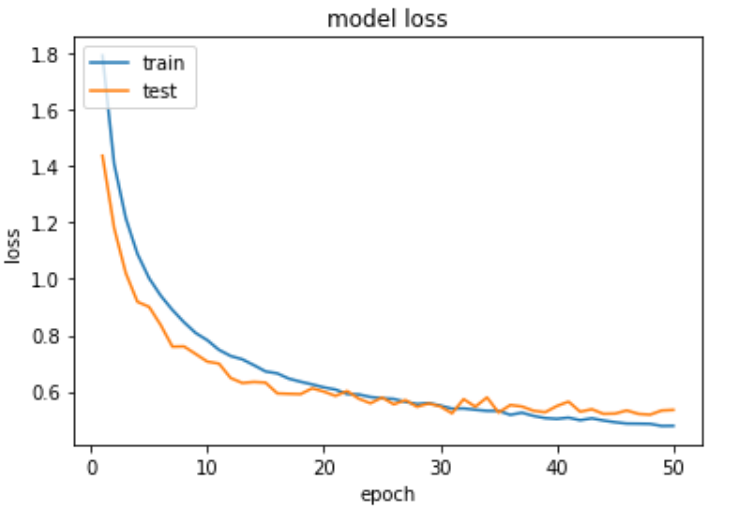
loss function은 학습을 통해 얻은 데이터의 추정치와 실제 데이터의 차이를 도출하는 함수로, 값이 크면 클수록 많이 틀렸다는 의미고, 이 값이 0이면 완벽하게 추정할 수 있다는 의미이다. loss function으로 categorical\_crossentropy를 사용하였다. 이 함수는 출력값이 one-hot encoding인 경우에만 사용할 수 있다. 즉, 답이 0또는 1이며, 이것이 백터 형태, 예컨데 [0,0,1] 등의 형태인 경우에만 사용할 수 있는 것이다. 특히 이 categorical\_crossentropy는 클라스가 3개 이상일 때 사용하는 crossentropy loss function이다. 함수는 다음과 같은데, 이때 y는 실제값 (0 또는 1) , 을 예측값 (확률)이다. 낮은 확률로 예측해서 맞추거나, 높은 확률로 예측해서 틀리는 경우 loss가 더 크다.





학습을 한 후 loss와 accuracy를 확인하면 다음과 같다. accuracy가 80%을 넘는 것을 확인할 수 있다.



또한 loss그래프는 다음과 같다. 약간의 overfitting 문제가 보이나 대체적으로 train과 test set의 loss가 같이 감소하고 있음을 볼 수 있다. epoch 35 정도부터 train과 test loss가 벌어지는 문제(overfitting 관련)는 Dropout의 크기를 더 크게 하면 해결될 문제이다.

* 1. 데이터 예측(Confusion matrix)

다음은 일부 데이터에 대한 예측 값을 보여준다. 한 개 데이터를 제외하고 정답을 맞힌 모습이다.



다음은 confusion matrix이다.

