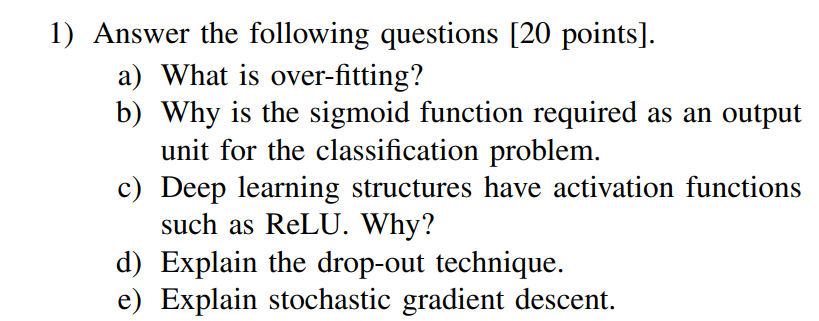
202388502

김명찬



1. Overfitting이란, 모델을 훈련할 때, train\_set에 최적화 되어, train에서의 성능은 높이 나오지만, test의 성능은 이에 비해 현저히 떨어지는 것을 말한다.

ex) train\_accuracy : 90%, test\_accuracy : 50%

1. sigmoid는 활성화 함수로서, 노드의 총합을 0~1 사이의 값을 나타나게 한다.

sigmoid는 특히 이진분류 classification을 할 때 유용하다. 따라서, Logistic Regression에서 주로 쓰인다.

예를 들어, 강아지 고양이를 분류한다고 하자. 훈련시킨 model에, Test를 한다면, 강아지 사진을 넣었을 때, output 직전 sigmoid 함수를 걸치게 되면, 강아지와 고양이 각 각의 label 확률을 0에서 1 사이의 값으로 예측하고 그 확률에 따라 가능성이 더 높은 범주에 속하는 것으로 분류해준다. 이와 같은 예에는 강아지 label에 높은 확률이 나오는 것이 좋은 모델이다.

1. **Activation function**은 이전 층(layer)의 결과값을 변환하여 다른 층의 뉴런으로 **신호를 전달하는 역할**을 한다. 활성화 함수가 필요한 이유는 모델의 복잡도를 올리기 위함인데, 특히 **비선형 문제를 해결하는데 중요한 역할**을 한다.

하지만 기존에 나왔던 활성화 함수는 기울기 소실(gradient descent) 문제점이 있었다. 이를 해결하기 위해서 나온 함수가 ReLU이다.

ReLU는 미분을 해도 값을 소실하지 않았고, 입력값이 양수인 경우만 뉴런을 전달하였다. 또한, 속도가 빠르다는 장점이 있다. .

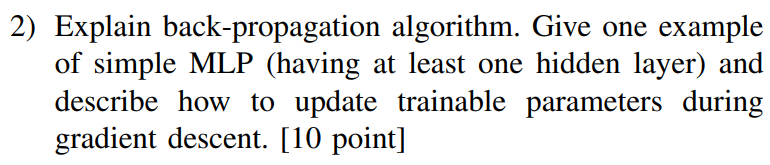
1. Dropout은 outfitting을 해결하기 위해 고안된 방법이다.

다차원 층(layer)을 가지는MLP는 overfitting 문제점이 자주 등장했다. 이를 해결하기 위해, train시에 의도적으로 랜덤적으로 선택되는 노드를 의도적으로 삭제하여 문제를 해결하는 방법이다.

1. SGD(Stochchastic gradient descent)는 GD(gradient descent)의 문제점을 개선한 방법이다.

GD는 train시에 1epoch 당 전체 데이터를 한 번에 연산하기 때문에, 연산량이 많아져 속도가 느려지고, 메모리에도 큰 부담감을 주었다.

하지만, SGD는 1epoch 당 전체데이터 중에서 랜덤적으로 선택된 데이터만을 이용하였기 때문에 속도와 메모리 문제를 해결하기 위해 고안된 방법이다.



역전파(Back-propagation)는 target 값과 모델이 계산한 output이 얼마나 차이가 나는 지 구한 후 그 오차값을 다시 뒤로 전파해가면서 각 노드가 가지고 있는 변수들을 갱신하는 알고리즘이다.

변수들을 갱신하는 방법은 각 노드의 미분 값을 이용하여 weight들을 증가시켜야 할 지, 감소시켜야 할지 결정한다.

ex) Input Node(X) – (w1 = 1)-> Hidden layer(Z) with ReLU activation – (w2 = 2)-> Output Node(Y) with ReLU function

input = 1, output = 2, target Y = 3, learning\_rate = 1

Backpropagation pipeline

1. **error 값을 구한다.**

* Z = ReLU(w1 \* X) = 1 -> Y = ReLU(w2 \* Z) = 2

Y값과 실제값(target)을 이용하여, loss function을 이용하여 error 값을 구한다.

MSE를 이용해서 구한다면, (3 – 2) \*\* 2 = 1

1. **Backpropagation**

* Loss 값의 기울기를 Chain rule을 이용하여 구한다.

순서대로 구해보자면,

1. dLoss/dW2 = (dLoss/dY) \* (dY/dW2)

dLoss/dy = 2 \* (Y-target), dy/dW2 = H

그래서, dLoss/dW2 = 2 \* 1 \* 1, 즉 2이다

1. dLoss/dW1 = (dLoss/dY) \* (dY/dH) \* (dH/dW1)

dY/dH = W2, dH/dW1 = X

그래서, dLoss/dW1 = 2 \* 1 \* 2 \* 1 = 4

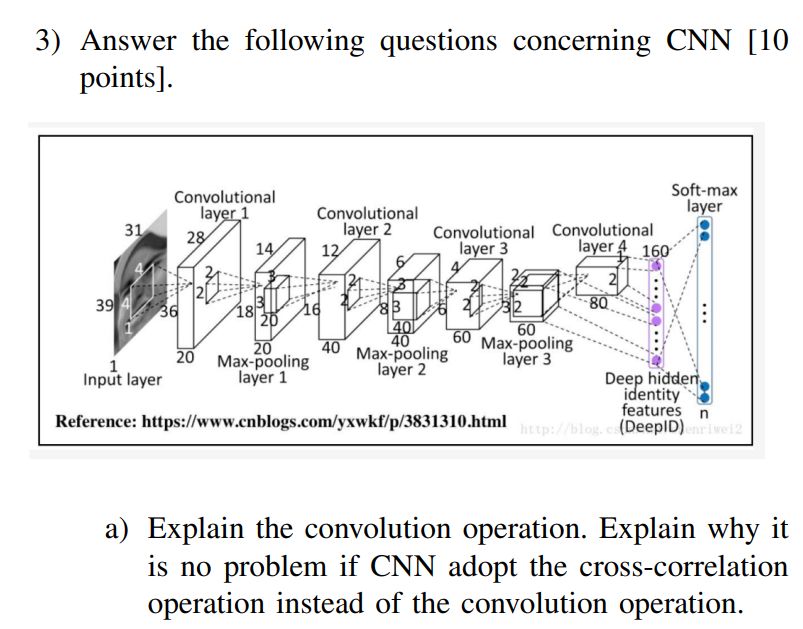
1. **파라미터 갱신**

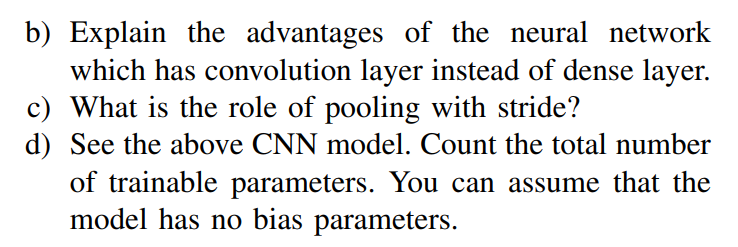
* W1 and W2 값을 계산된 기울기와 learning rate 값을 이용하여 갱신한다.

W2 = W2 – learning\_rate \* dLoss/dw2

W1 = W1 – learning\_rate \* dLoss/dw1

W2 = 2 – 1 \* 2, W2 = 2, W1 = 1 – 1 \* 4, W2 = -3 으로 weight들이 갱신될 것이다.





CNN은 주로 이미지에 쓰이는 알고리즘이다. 따라서 밑의 예에서 input 값을 이미지라고 가정하고 설명하겠다.

1. 합성곱 신경망의 입력값에 필터를 곱할 때, 필터를 뒤집어서 적용한다. 하지만, Cross-corrleation은 뒤집지 않고 필터를 곱한다. Train 의 목적은 필터의 값을 곱하는 데에 목적을 가지고 있다. 뒤집지 않을 뿐 값이 누락되는 것이 아니기 때문에 Cross-corrleation을 사용해도 문제가 되지 않는다.
2. Dense layer는 종종 데이터의 형상이 무시된다. 예를 들어, 이미지 데이터를 input 값으로 넣었을 때 경우 생각을 해보자. 이미지는 (28,28,3) 즉, 3차원 값이다. 하지만, Dense layer를 통과할 때는 3차원 형태가 아닌 1차원으로 flatten 작업을 겹치게 된다.

하지만, 3차원 공간에는 많은 정보가 들어가 있다. 가까운 픽셀은 값이 비슷하거나, RGB의 각 채널은 서로 밀접하게 관련되어 있거나, 거리가 먼 픽셀끼리는 연관성이 비교적 떨어진다. 합성곱 층 (convolution layer)에서는 이러한 정보를 유지한 채 train 된다는 장점이 있다.

1. Stride는 필터가 입력 이미지에서 많이 이동하는지를 결정하는 역할을 한다. 보폭이 클수록 각 단계에서 더 많은 픽셀을 건너뛰기 대문에 출력 feature map이 작아진다. 반대로 보폭이 작을수록 각 단계에서는 feature map이 커진다.

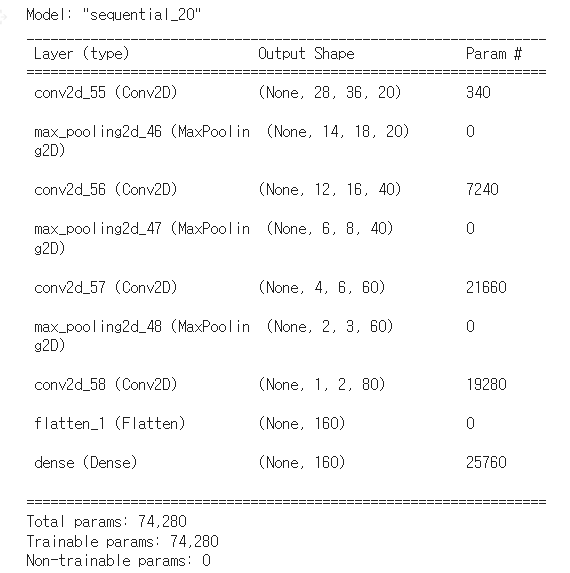
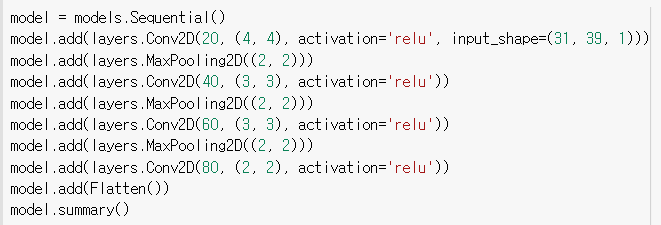
즉, 해상도의 차이점을 줄 수가 있다. Feature map이 작으면 해상도가 낮아지고, 커지면 해상도가 높아진다.

Train 시 해상도가 낮으면 underfitting의 위험이, 반대로 높으면 overfitting의 위험성이 있다. 그래서 stride는 적절히 조정하고 사용해야 하는 hyperparameter 역할을 한다.

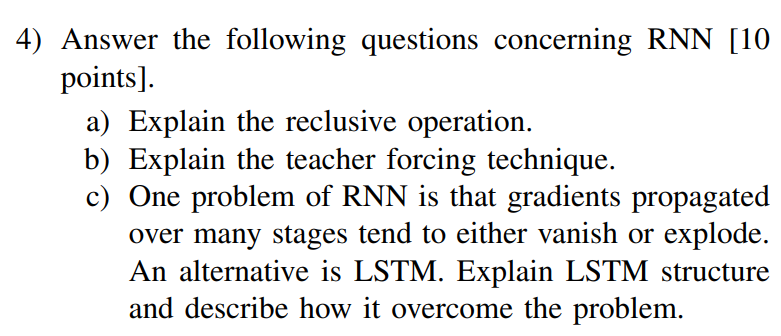
Pooling은 중요한 특징을 보존하면서 입력 특징 맵의 공간 차원을 줄이는 일종의 다운 샘플링 작업이다. 일반적으로 특징 맵의 공간 해상도를 낮춰, 모델의 복잡성을 제어하기 위해 쓰인다. 즉, overfitting을 막고, 연산 속도를 빠르게 하는 장점을 가진다.

종류에는 최대 풀링과 평균 풀링이 있다. 최대 풀링은 feature map에서의 가장 중요한 특징을 유지하는 데 도움이 되고, 평균 풀링은 노이즈를 줄이는 역할을 한다. 이미지에서는 주로 최대 풀링이 많이 쓰인다.

풀링 크기 또한 사용자가 설정할 수 있는 hyperparameter 역할을 한다.



사진에는 마지막 softmax에 적용될 class 개수가 나와있지 않았다. 만약, 160개를 한다고 가정했을 때, 총 계산할 parameter 개수는 7428이지만, 합성곱만을 계산한다면, 48520 개이다.



1. RNN은 입력층에서 은닉층으로 연결된 네트워크 안에 자기 자신에게 되돌아오는 루프를 가지고 있다. 이를 relusive operation 과정이라 칭한다.

Input(xt) –w1- > hidden layer(h) –w2-> output(yt) 구조를 가진다고 가정해보자.

h는 y에게 보내는 역할과 현재 시점 이전인 t-1 값을 보존하는 역할을 한다. 만약, 위 모델을 2번 반복한다고 가정해보자. 그리고 2번째 상황이다.

yt값은 w1, w2 한 시점 이전인 ht-1 가중치도 곱해질 것이다.

1. RNN a번에서 설명한 방식대로 진행된다. 하지만, 초기 output 값이 잘못 나오게 되면 전체 값이 틀리게 된다. 이러한 단점을 막기 위해 고안된 방법이 ‘teacher forcing’ 방법이다.

Teacher forcing은 RNN처럼 ht-1 값을 사용하는 것이 아닌 실제 정답 레이블을 이용한다. 이와 같은 방식을 이용하게 되면, 정확성이 올라가고 연산량이 줄어들기 때문에 속도가 빨라진다.

1. RNN은 루프를 반복하다 보면 신호가 점점 약해져 결국에는 소멸되는 문제가 발생할 수 있다. 이런 점을 극복하기 위해 고안된 방법이 LSTM이다.

LSTM의 핵심은 cell state이다. 정보를 전혀 바뀌지 않고 그대로 흐르게 하는 것을 수행한다. 이에 정보를 추가하거나 없앨 수 있는 게이트들이 있다.

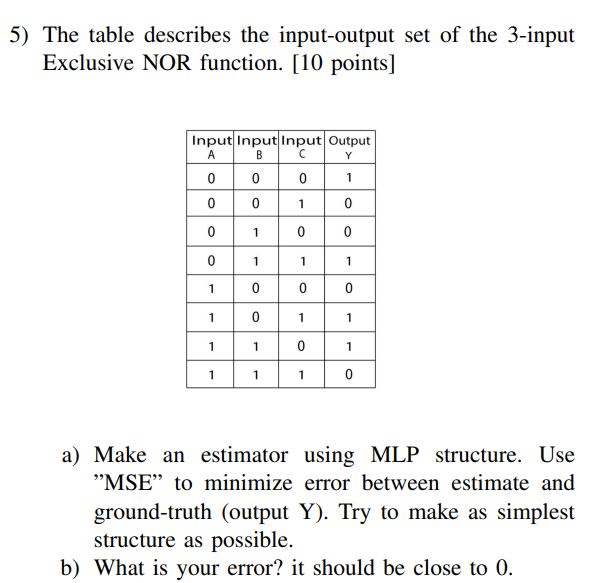
망각 게이트, 입력 게이트, 출력 게이트 등이 이에 해당한다

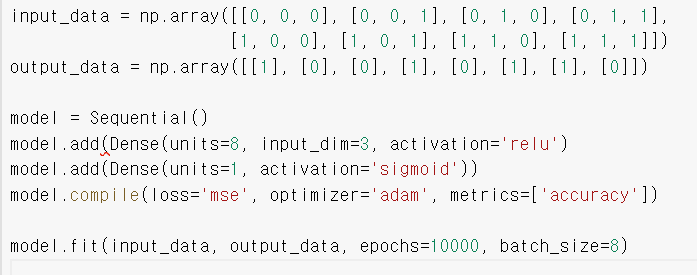
망각 게이트: 과거 정보를 얼마나 잊어야 하는지 결정한다. 현재 input 값xt와 과거 cellstate 값 ht-1의 값들을 입력으로 받아 sigmoid를 통과시킨 후 메모리 셀의 각 요소에 대해 0과 1 사이의 출력합니다. 만약, 0이라면 이전 상태의 정보는 잊고, 1이라면 정보를 완전히 기억하게 된다.

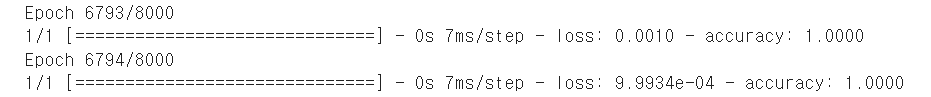
입력 게이트 : 현재 정보를 기억하기 위한 게이트다. 값을 얼마나 cell state에 반영할 것인지를 결정한다. 우선, 망각 게이트와 마찬가지로 sigmoid를 통해 어떤 값을 업데이트할지 결정한다. 다음, tanh을 통해 출력값으로 반환되서 cell state에 더해진다.

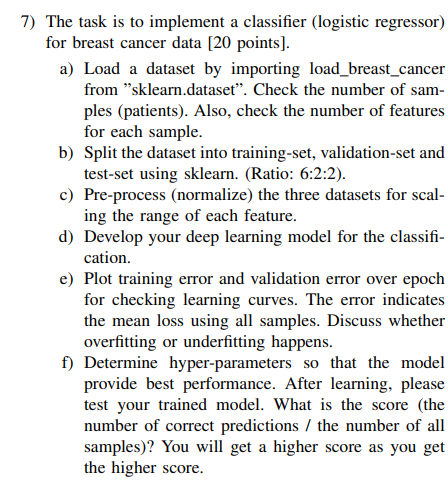
출력 게이트: 최종적으로 얻어진 cell state의 어느 부분을 output으로 내보낼 지 결정한다. 우선, 최종적인cell state를 sigmoid에 통과시켜 어느 부분을 출력시킨지 결정한다. 그 후, tanh을 통과한 값이 출력된다. 또한 이 출력된 값은 ht 값과 곱해져 다음 cell state에 들어간다.

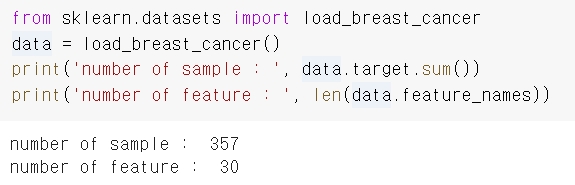
위와 같은 과정을 반복한다.

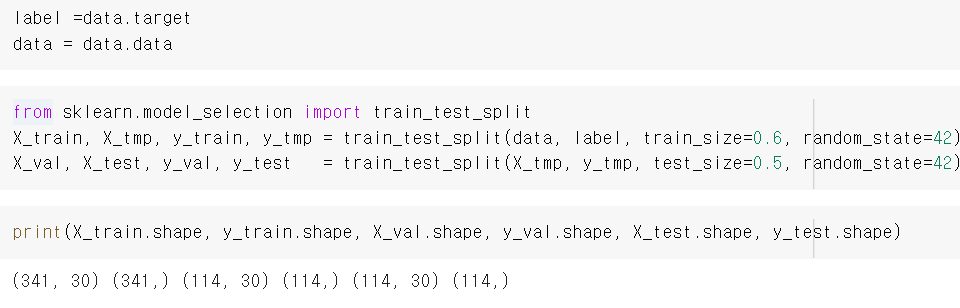


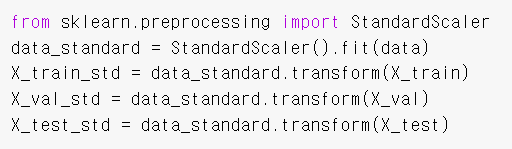


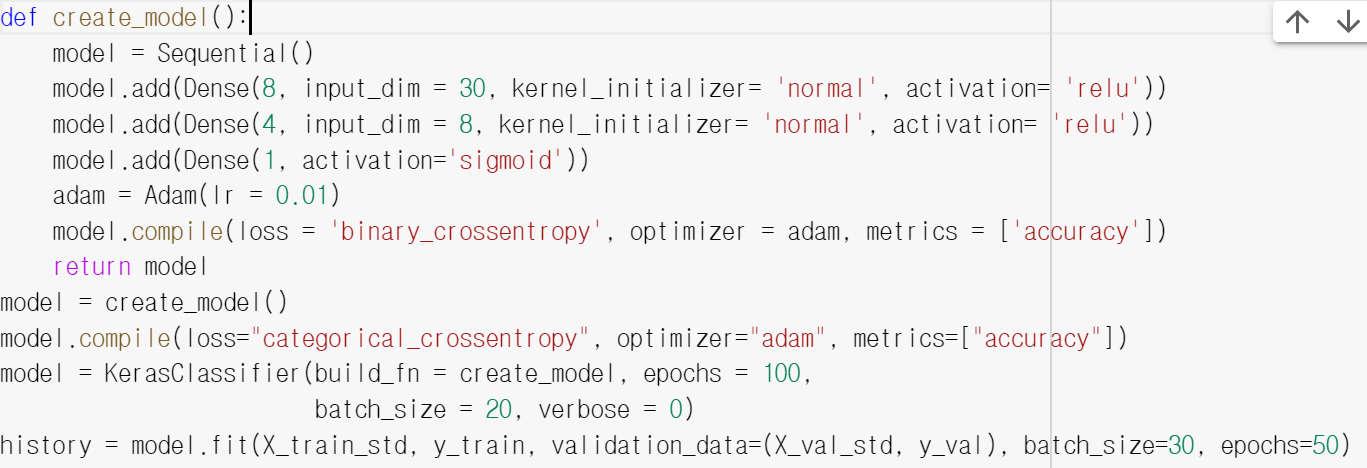


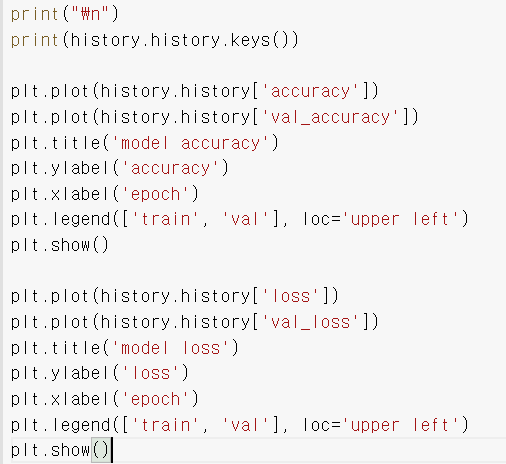
1. 

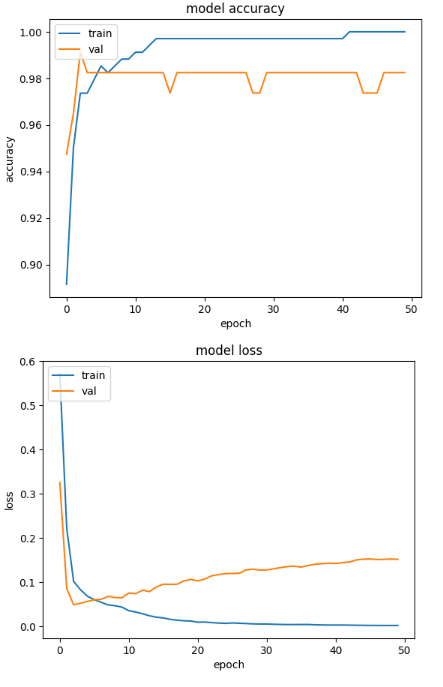






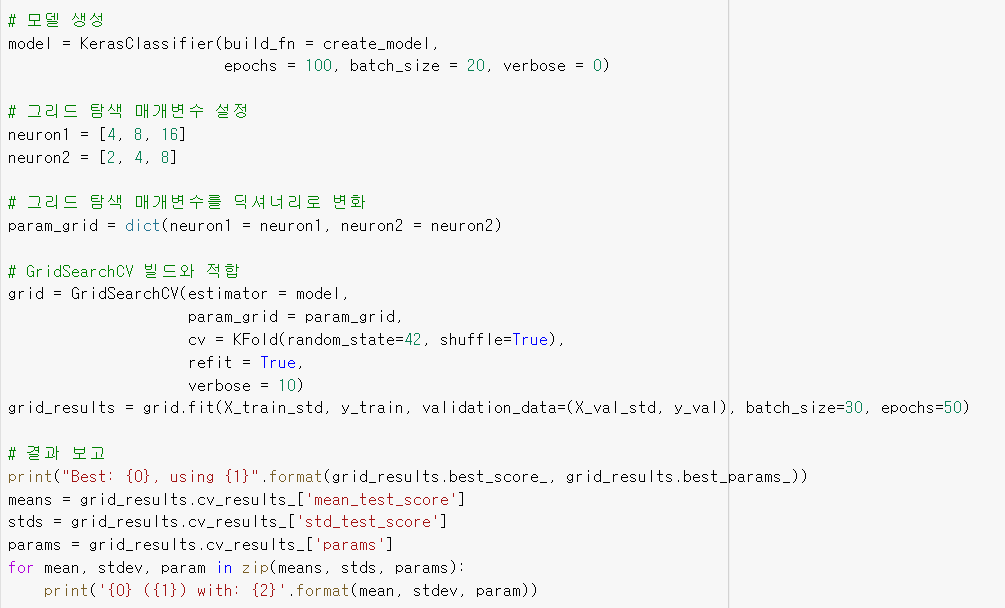
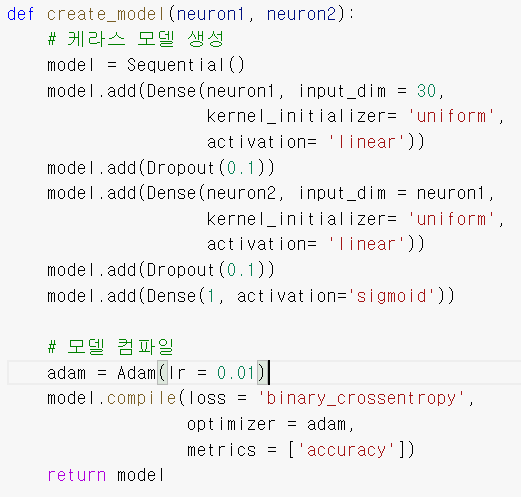




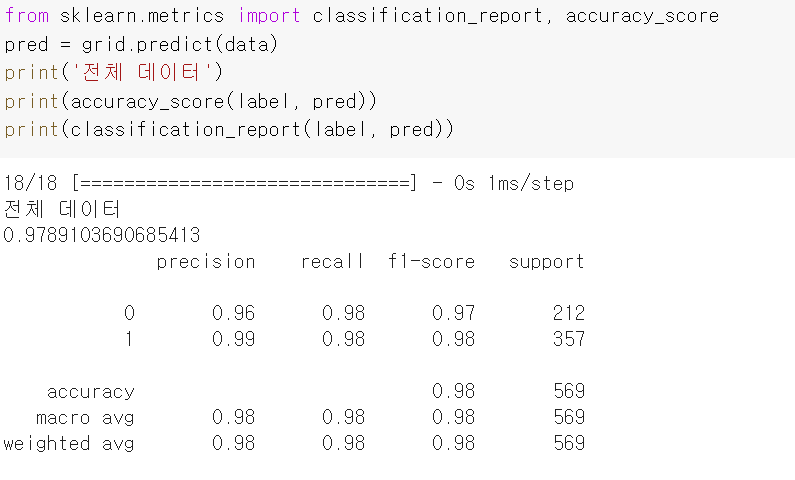


train\_accuracy 값과, validation 값은 둘 다 정확성이 1에 가깝기 때문에 underfitting, overfitting 둘 다 해당되지 않는다. 단, epoch가 진행됨에 따라 미묘하지만loss가 늘어나는 것으로 보아, overfitting 가능성이 있다. (지금도 물론 좋은 모델이라고 판단된다)

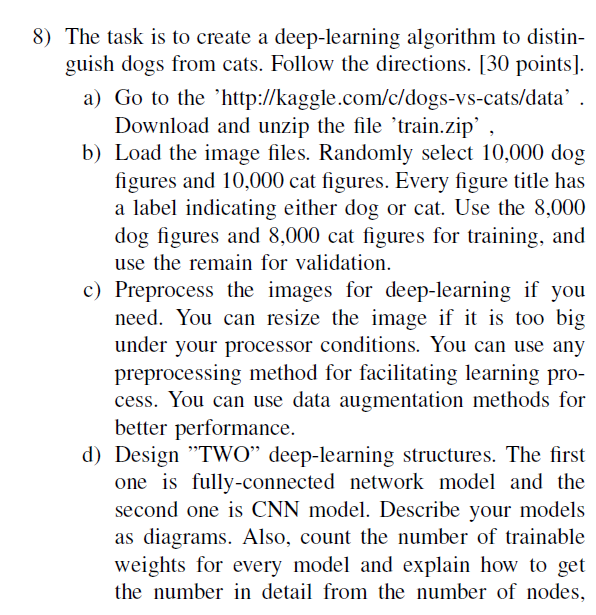
하지만 약간의 overfitting이 있다고 판단 됐으니, dropout 기술을 추가하겠다. 또한, 차원의 값을hyperparameter로 설정하겠다. 또한 편하게 찾기 위한 gridsearch 함수를 이용하여 적절한hyperparameter 값을 찾아보겠다.







전체 데이터를 넣었을 때, 정확도가 꽤 높게 나왔다.



8번과 9번은 같이 보낸 ipynb 파일 꼭 확인 부탁드립니다 !