**Machine Learning & Data Mining, Spring 2020**

**Homework 10**

Due June 19

**과제목표**

* **Backpropagation을 이해하여 구현할 수 있다.**
* **CNN을 이해하여 학습시킬 수 있다.**

**제출**

* **요구한대로 작성한 보고서와 주피터 노트북**
* **주어진 주피터 노트북을 바탕으로 요구하는 코드를 구현 후 보고서에 해당 부분을 캡처**
  + **요구하는 코드만을 캡처하여 보고서에 제출**
* **요구한 코드 이외에 다른 코드는 수정하지 말것.**
* **최종 점수는 비율적으로 수정해서 들어갈 예정**

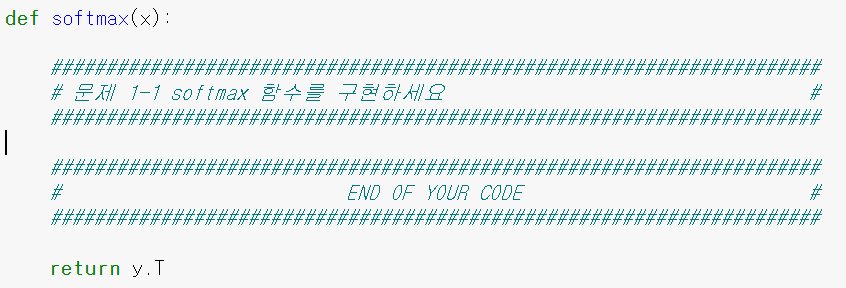
**학번: 201520908**

**이름: 유성민**

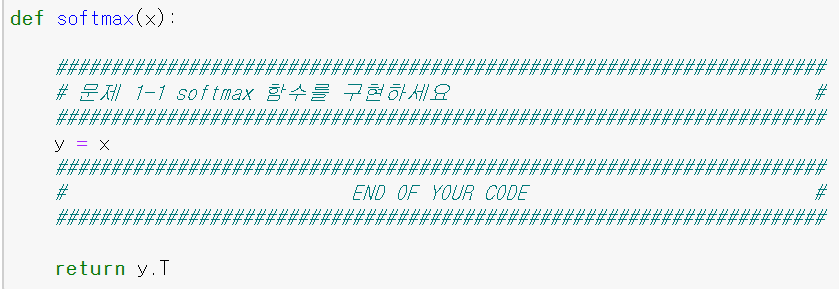
**학년: 3**

**학과: 소프트웨어**

**보고서 코드 작성 예시**

****

문제 1-1 번의 경우 다음과 같이 문제가 주어집니다. 제가 작성한 주피터 노트북의 최종 학습을 할 수 있도록 코드를 작성한 합니다. 예를 들면 다음과 같이 코드를 작성한 후에,



작성한 부분을 캡처 하시면 됩니다. 특히,

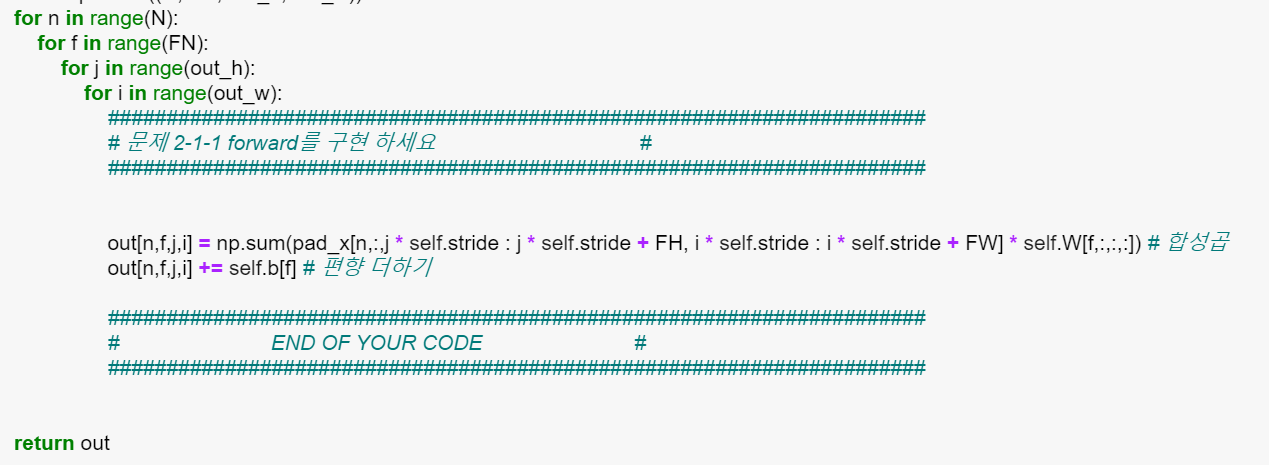
* 워드 글자와 캡처한 그림의 글자 크기가 유사하도록 캡처 하고
* 실제 워드로 코드내용을 적을 경우 **가독성이 떨어지므로** 되도록 캡처합니다.

어떠한 방식으로 작성해야 되는지에 대한 힌트는 주피터 노트북의 주석에 달아 놓겠습니다.

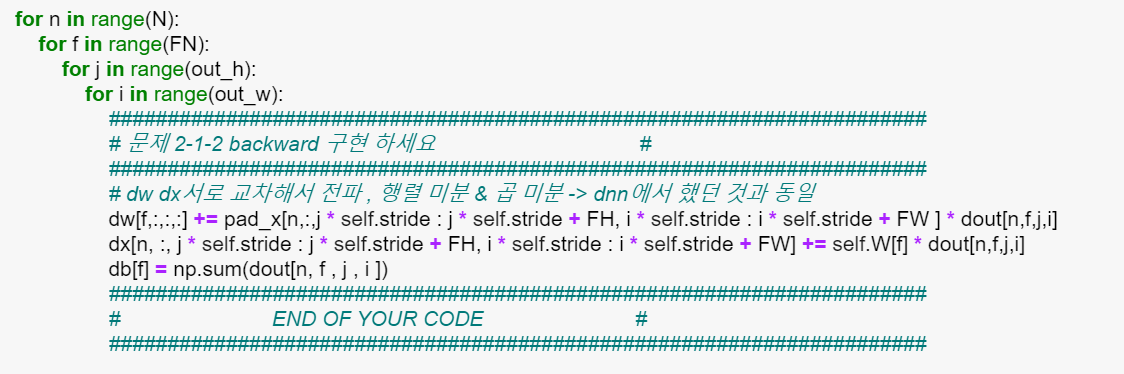
**주피터 노트북의 설명에 따라서 해당 문제의 코드들을 캡처하세요.**

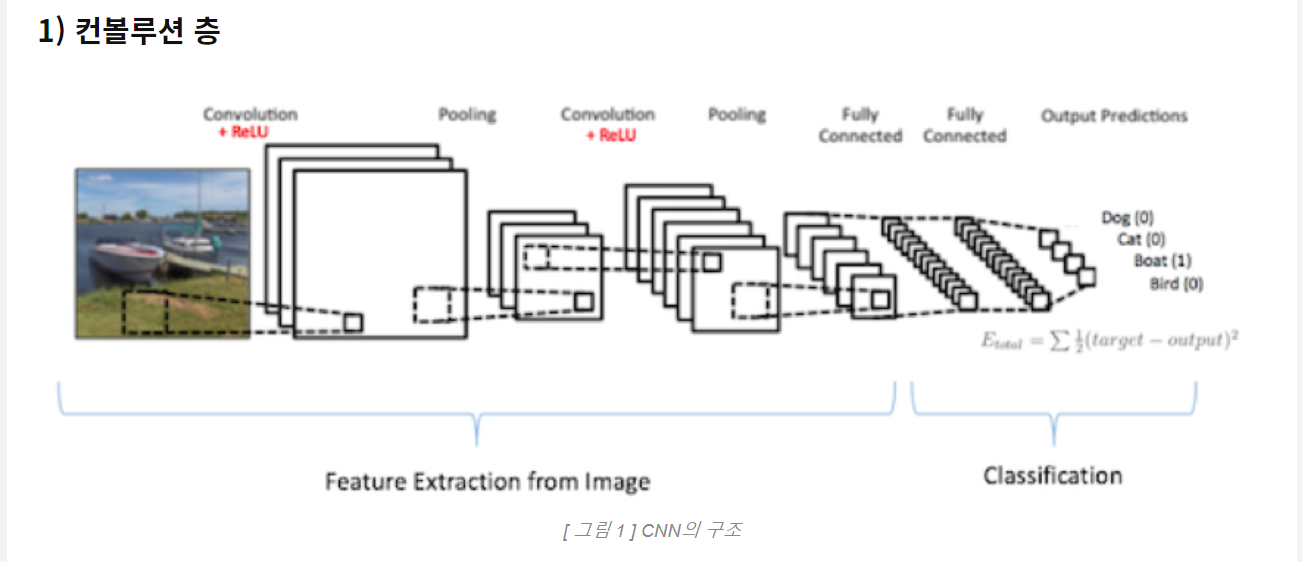
**[2] 주어진 코드를 바탕으로 CNN 학습 완성**

**2-1 convolution layer forward & backward 완성 (for문 사용)**

****

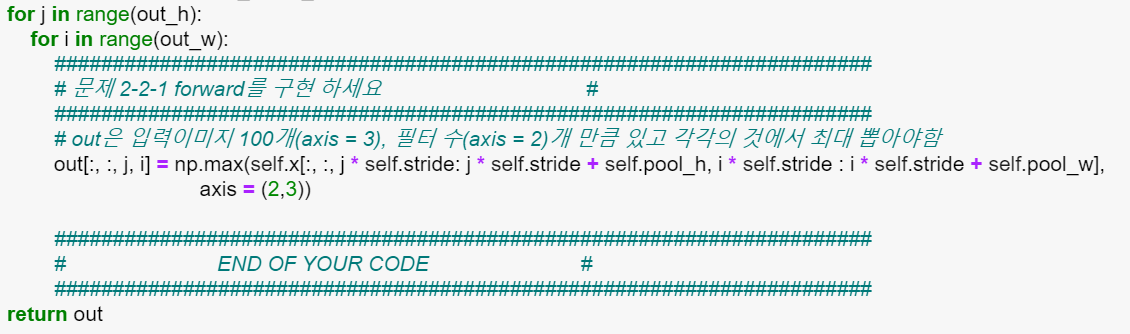
**<convolution layer forward - 합성곱 부분>**

****

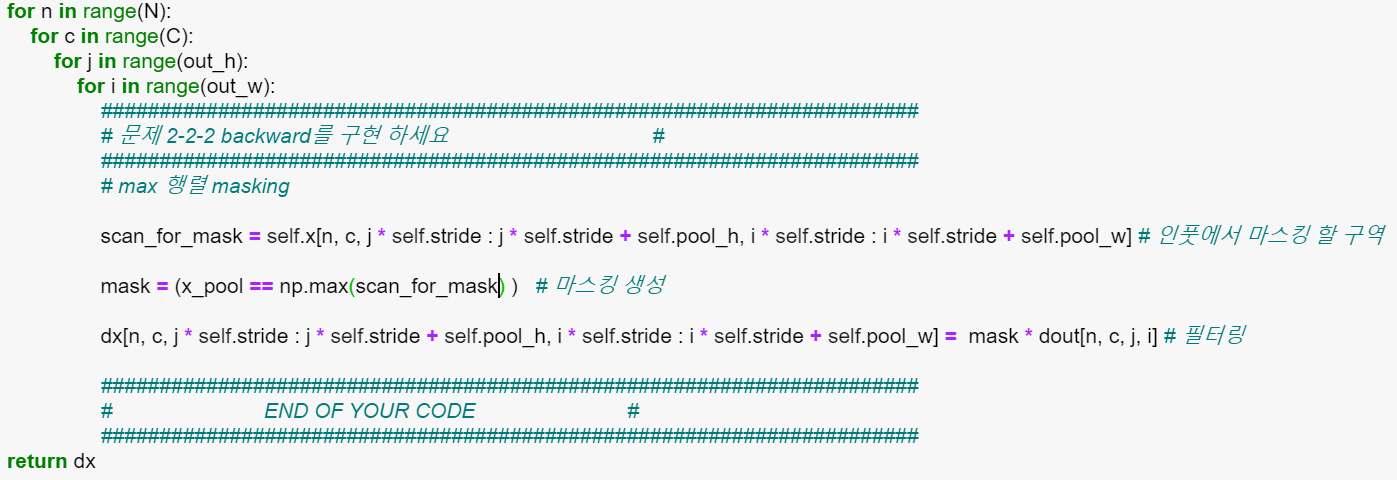
**<convolution layer backward - dw, db, dx 미분>**

**먼저 convolution입니다. forward에서는 합성곱연산만 해주면 되기때문에 input의 local area 즉, i : i + stride x j : j + stride 부분의 사각형과 필터 w를 곱하고 더합니다. 저 연산은 element level로 이루어 집니다. 그리고 bias를 더하고 forward는 끝납니다. 다음으로 backward입니다. backward는 이전과제에서 다룬바와 같이 행렬곱미분입니다. 그 공식을 보면 w와 x가 교차되어 전파됩니다. bias는 마찬가지로 열기준 합입니다. 이 두가지 과정을 거쳐서 convolution layer를 구현했습니다.**

**2-2 max pooling layer forward & backward 완성 (for문 사용)**

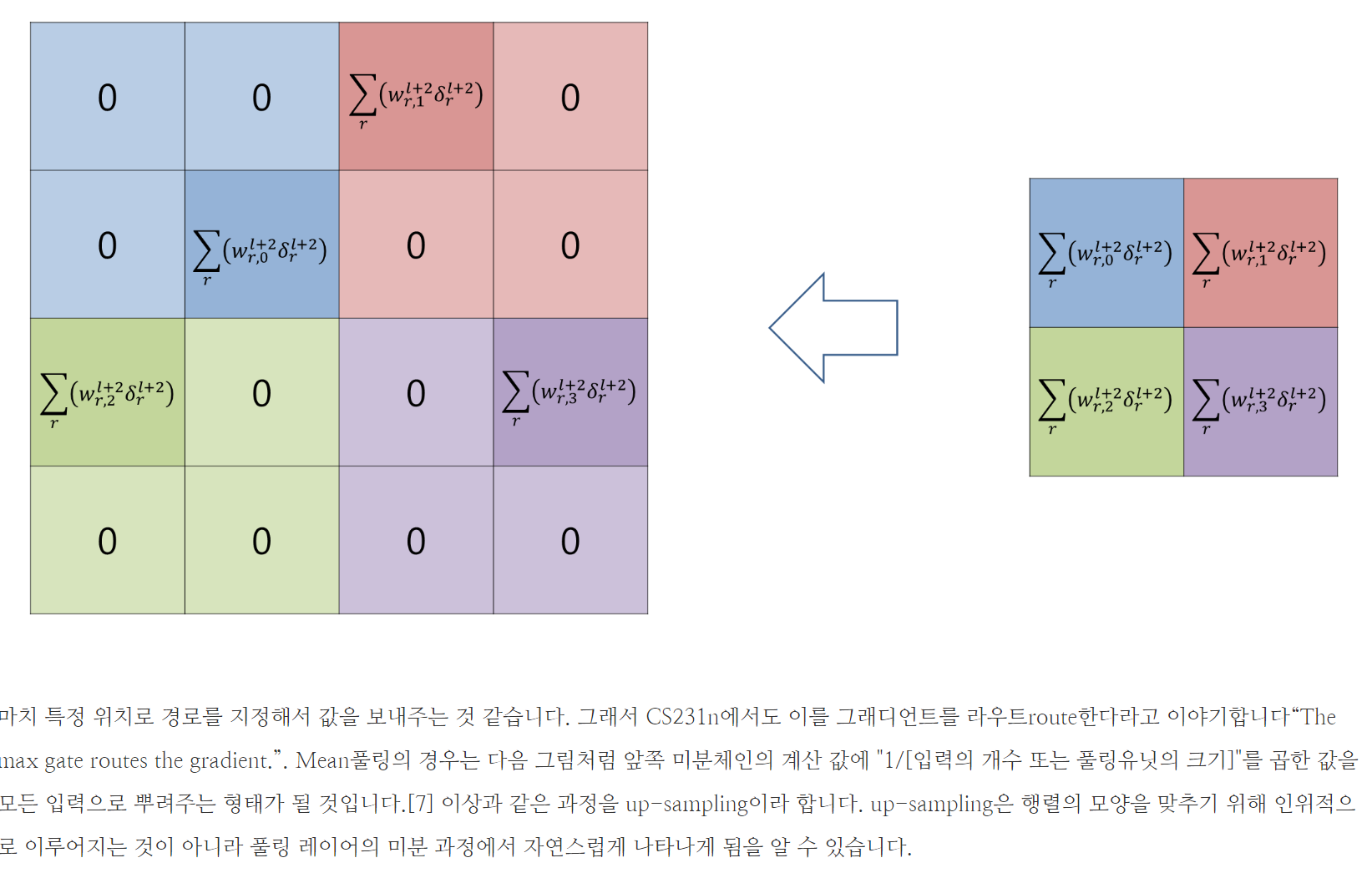
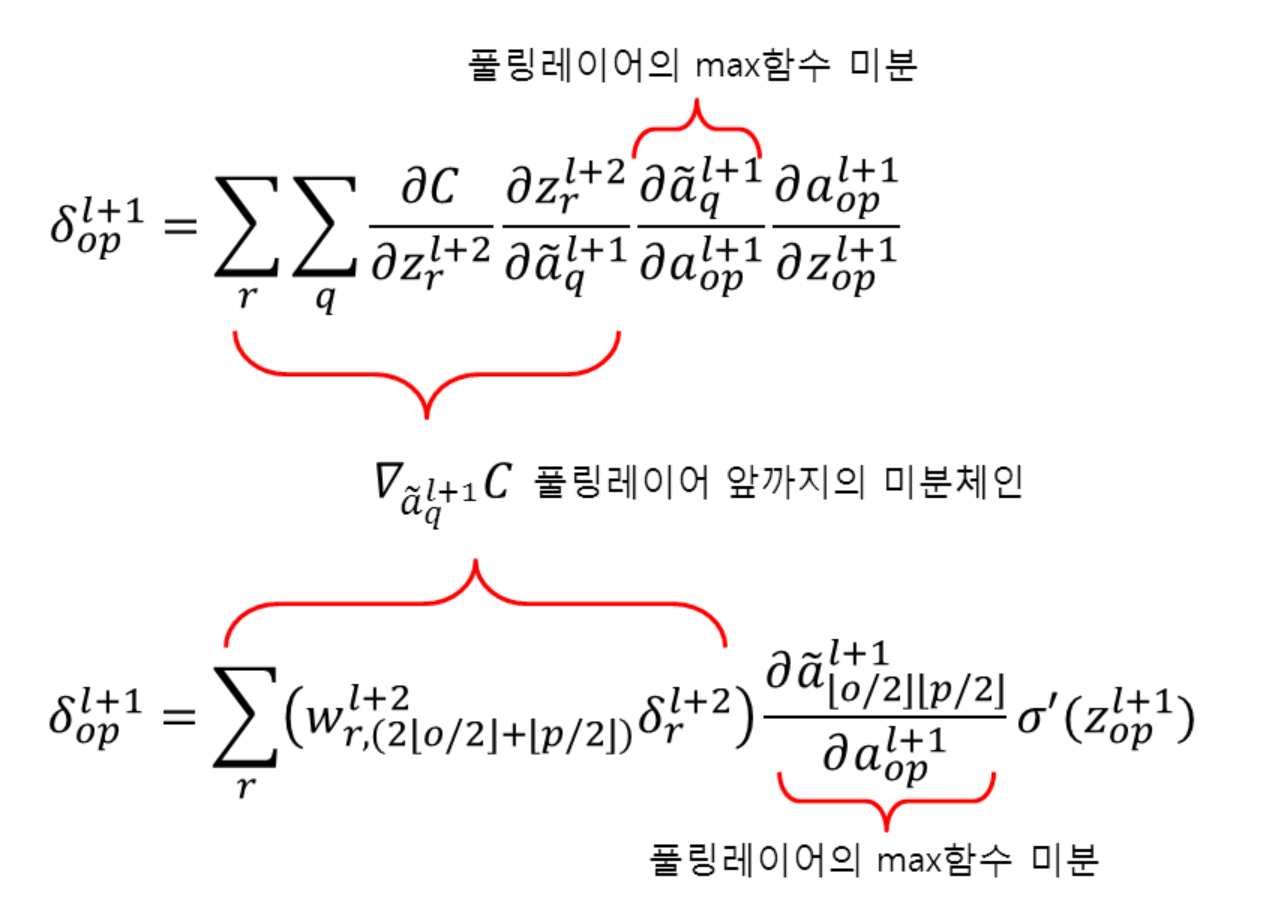
****

**<max pooling layer forward>**

****

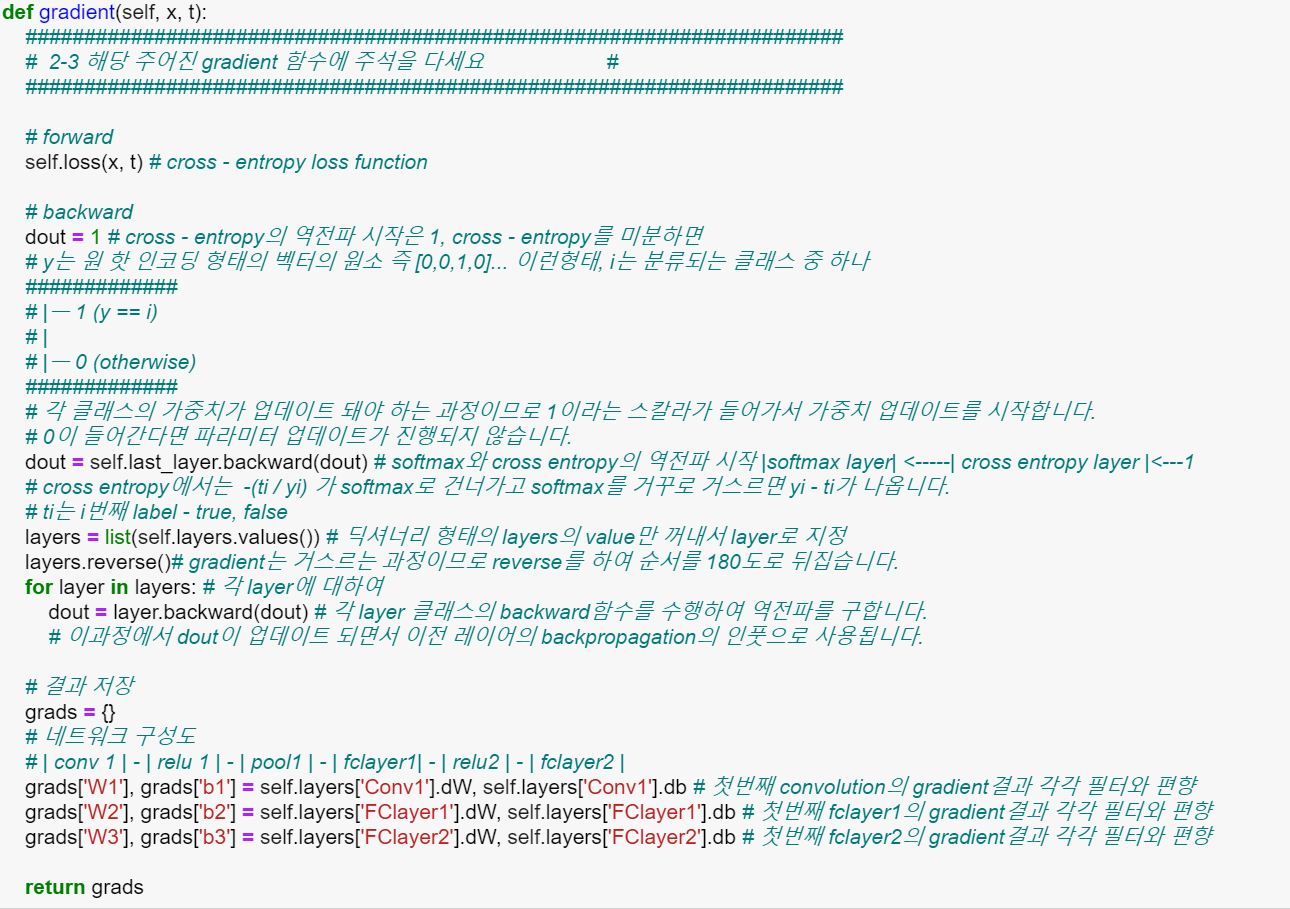
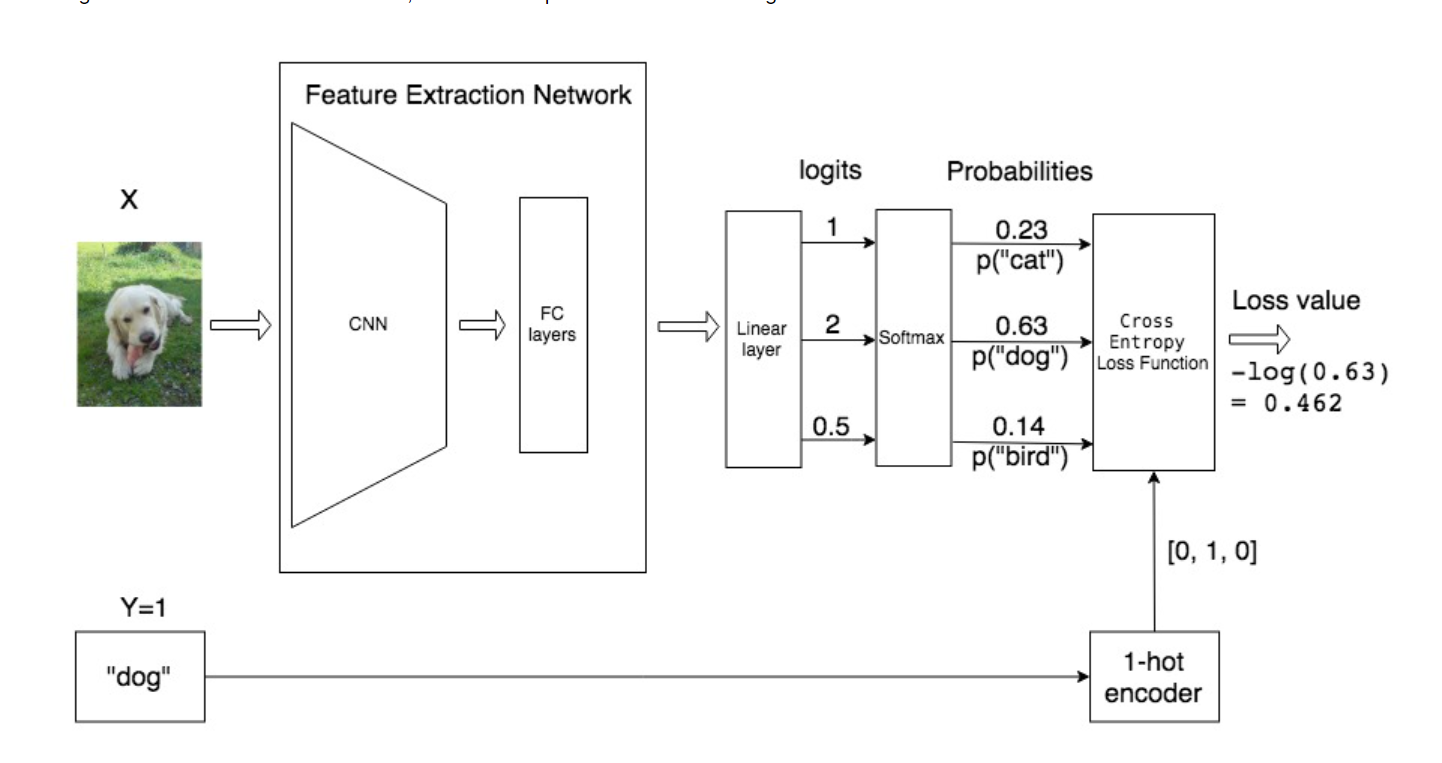
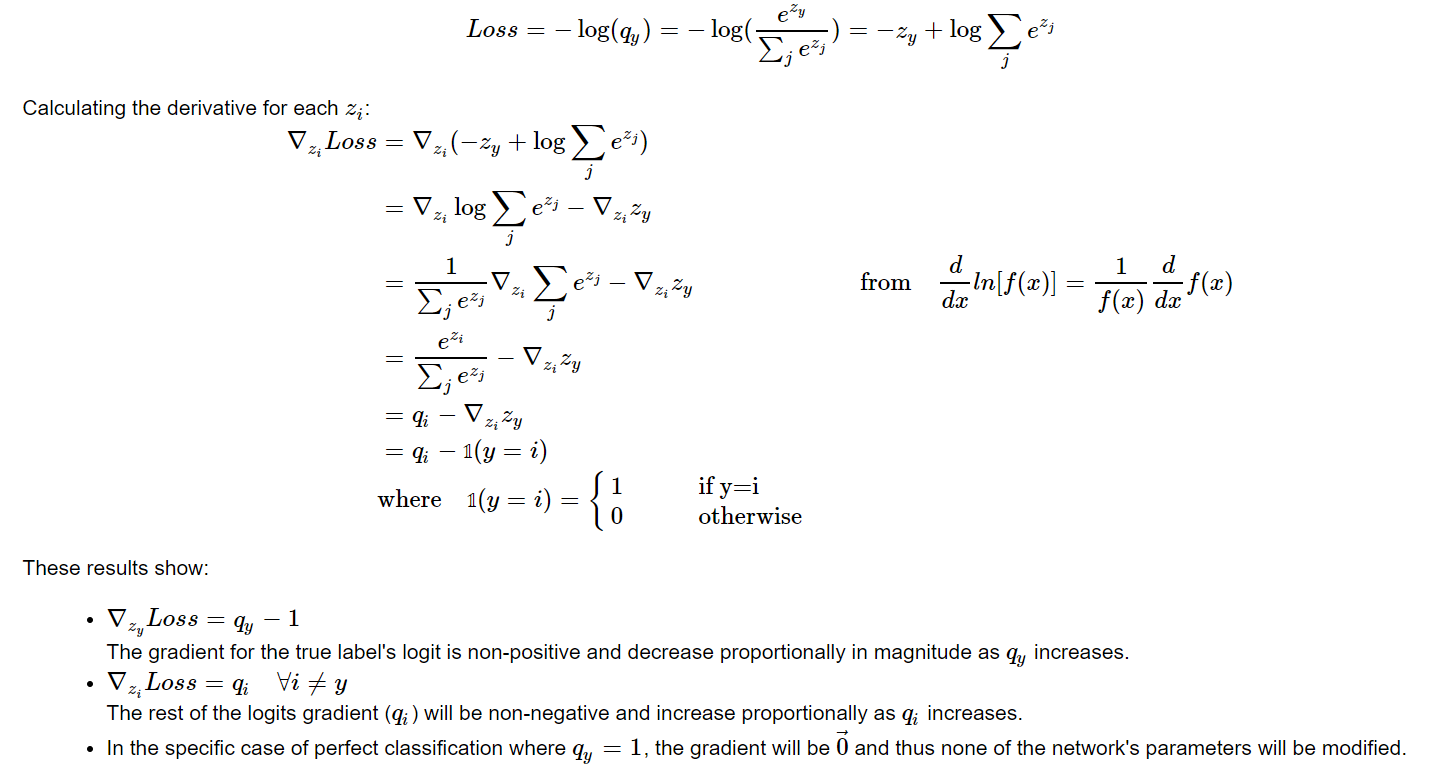
**<max pooling layer backward>**

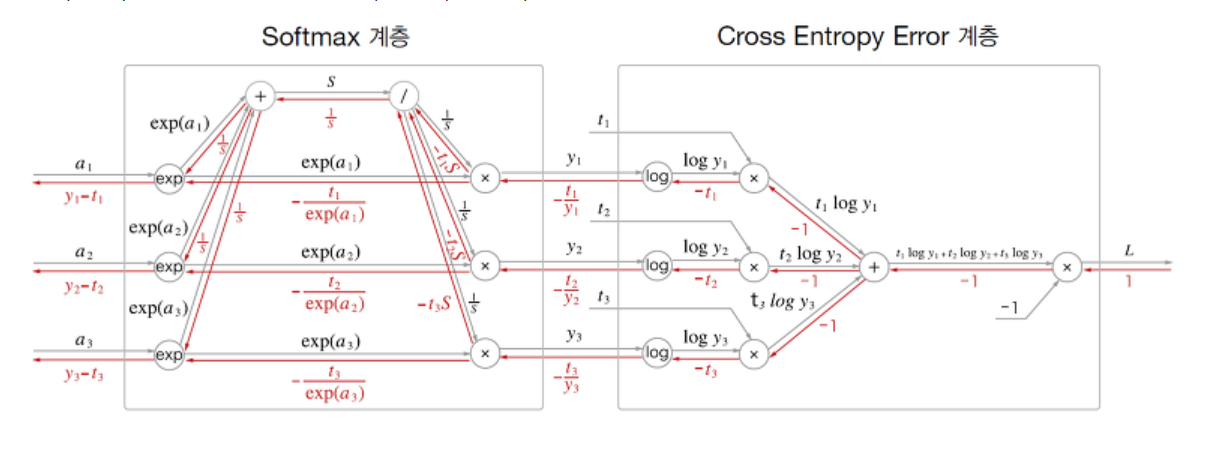
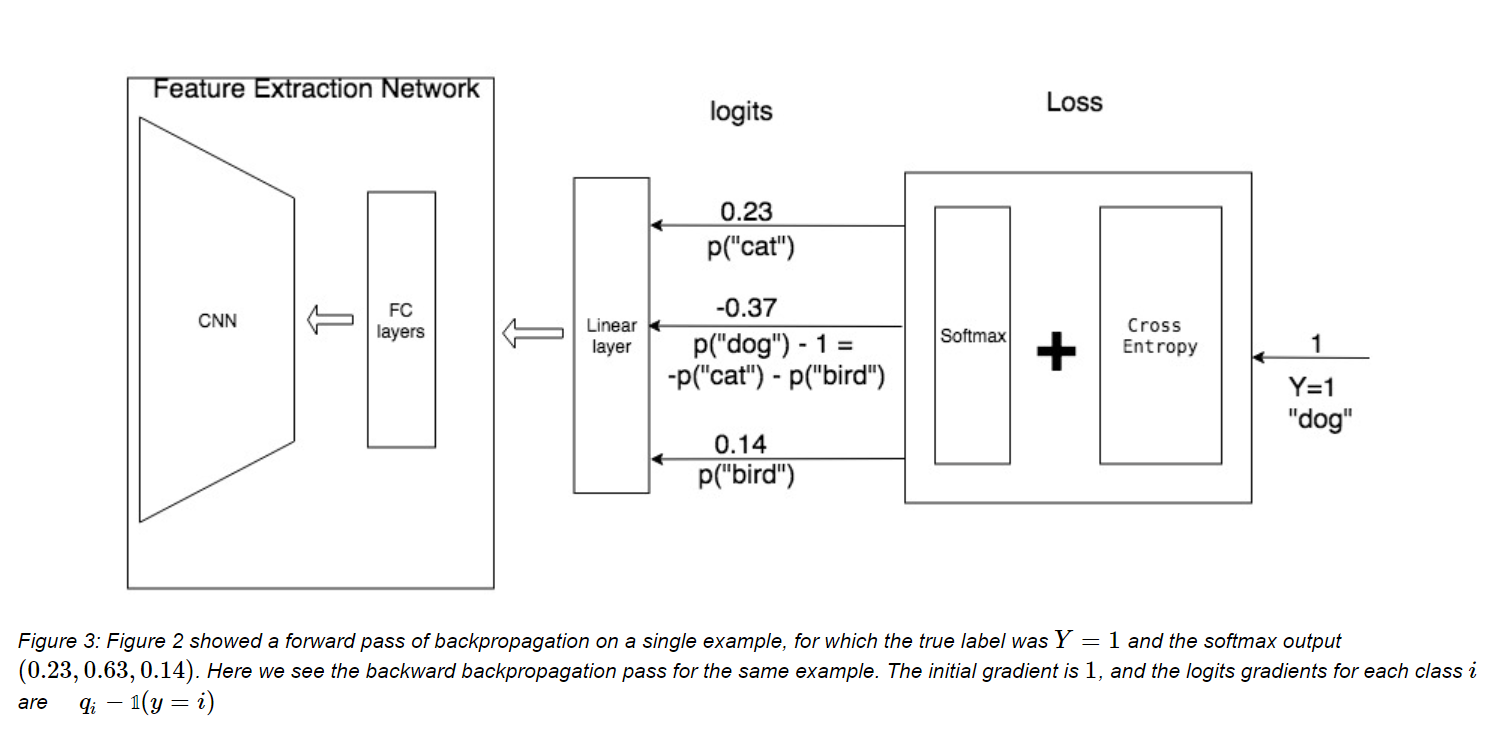
**maxpooling layer는 2x2로 이루어지기때문에 input의 크기가 n x m 이었다면 n/2 x m/2로 줄입니다. localarea에서 max값을 골라서 out에 저장시키는데 axis = 2,3을 기준으로 해야합니다. 왜냐하면, 2는 filter의 갯수를 의미하는 차원이고, 3은 input 갯수를 의미하는 차원이라서 위의 그림을 참고하면 max pooling을 거친 후에 output은 filter의 갯수만큼 나오고 그것이 input 갯수만큼 있어야 하기 때문입니다. 그래서 필터 별로 최대값 뽑고, 그 이후로 인풋별로 최대값을 뽑아야 맞습니다. 해당 과정을 생략하여 정확도가 너무 낮게 나와서 다시 공부한 끝에 정확도를 올렸습니다. 다음으로 backward입니다. 다음장에서 이어서 설명하겠습니다.**

****

**max pooling의 미분은 신기하게도 미분값을 원래자리로 routing해주는 과정입니다. 그래서 마스킹을 사용하면 구현이 간단해집니다. 나머지는 다 0이기 때문입니다. 일단 forward에서 사용했던 local area를 scan\_for\_mask에 저장하고 max 함수를 사용하여 max값의 위치만 True로 바꿉니다. 나머지는 다 0이 됩니다. 그래서 mask행렬이 생성되고 역전파가 올때, mask와 미분값을 담은 dout을 곱해주면 됩니다. 그러면 자연스럽게 max위치의 미분값만 필터링이 되어 다음 역전파로 건너갑니다.**

**2-3 주어진 네트워크의 gradient 함수에 주석을 써서 제출**

****

****

**다음은 cross entropy의 즉 loss함수의 미분과정을 나타낸 도식표입니다. dout을 처음으로 1으로 설정하는 이유부터 말씀드리자면, cross entropy의 미분결과는 0 또는 1인데 1이 나오는 경우는 i번째 클래스에 대해 y가 i를 예측했다면, 역전파에서는 미분값이 1이되고, 다른 부분에서는 0이 됩니다. 역전파의 의의는 어떤 이미지인지 잘 분류하기 위함입니다. 1을 넣게 되면, 모든 클래스에 대하여 필터값이 업데이트 되면서, 더 잘 필터링 될 수 있습니다. 만약 0을 넣게 되면 역전파에서 0이 거슬러 가면서 파리미터의 업데이트가 일어나지 않는 것을 보실 수 있습니다. 1이라는 스칼라를 넣음으로써 위의 도식표에서 y1, y2, y3 클래스가 업데이트 되는 과정을 볼 수 있습니다.**

**결과적으로 파라미터 업데이트를 위해서는 dout = 1로 설정해야합니다. 그 이후의 코드에 주석을 달아보자면, dout을 last\_layer에 넣고 대입하는데, 이 코드를 거치면서 softmax ce layer를 통과한 dout이 됩니다. layers라는 변수는 초기화 할 때 선언한 layers 딕셔너리의 value 즉, layer객체들을 뽑아서 리스트형태로 저장하고 이것을 reverse하는데, 초기화시에 forward 방향으로 저장했기 때문에, backward에서는 반대방향이 돼야합니다. 그리고 for loop을 돌면서 각 layer에 대해서 backward함수를 사용하며 dout을 업데이트 하는 과정이 일어납니다. 마지막은 각 결과값을 grad 딕셔너리에 저장하는 과정입니다.**

**2-4 주어진 코드로 학습을 한 후 학습된 필터를 출력 후 제출**

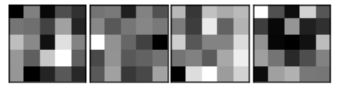
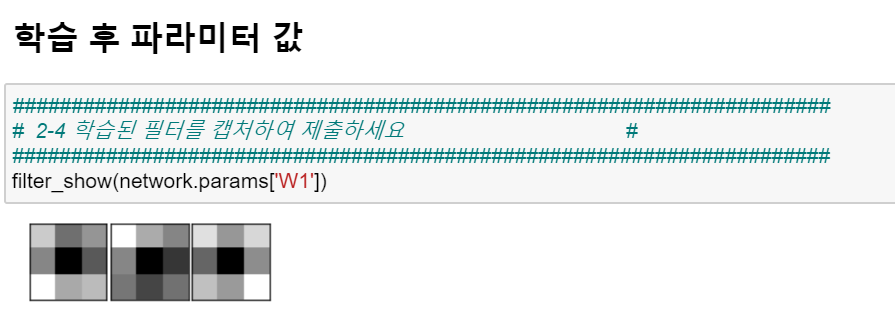


그림 1 예시 필터 실제 필터의 개수와 숫자 다름

****

****

****

****

**학습 후의 필터와 acc입니다. 처음에 max pooling을 잘못 코딩하는 바람에 0.113정도가 나왔습니다. 하지만 shape와 차원을 다시 공부하고 pooling을 수정하여 다시 돌려본 결과 다음과 같이 성능향상과 함께 학습된 필터를 볼 수 있습니다. 저는 해당과정의 진행사항을 보기위해 batch\_idx를 같이 출력하면서 보았습니다. epoch1에서는 0.775 다음 epoch에서는 0.87 마지막 epoch에서는 0.92정도의 acc가 나왔습니다. 필터도 골고루 잘 학습된 것을 볼 수 있었습니다.**

**-감사합니다-**