합성곱 신경망 (CNN)

2018.07.05

|  |  |  |
| --- | --- | --- |
| 작성자 | 검토자 | 승인자 |
| 이가영 |  |  |

© Copyright 2018 Medicisoft Co.,Ltd.

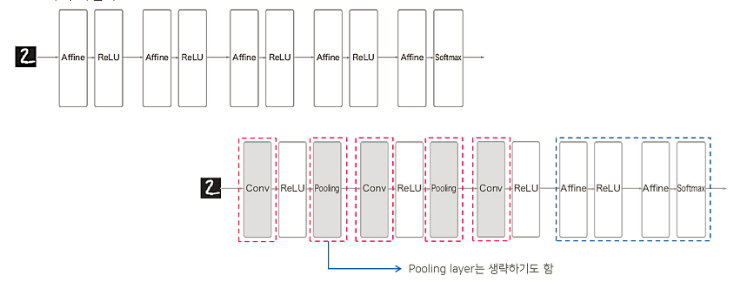
[1. 합성곱 신경망 (CNN – Convolutional Neural Network) 2](#_Toc518660179)

[2. 참고문헌 6](#_Toc518660180)

# 합성곱 신경망 (CNN – Convolutional Neural Network)

##### **구조**

* CNN의 구조는 완전연결계층(Fully-Connected Layer)과는 다른 점이 있다. Fully-Connected Layer에서는 Affine 계층으로 구현했지만, CNN에서는 아래의 그림 1과 같이 합성곱 계층(Convolutional Layer)과 폴링 계층(Pooling Layer)가 추가된다.

  
 그림 1) FC와 CNN의 구조적 차이

##### 기존 Multi-Layer Neural Network(MLNN)의 문제점

* 기존 MLNN의 문제점은 변수의 개수, 네트워크 크기, 학습시간 세 가지 문제가 발생할 수 있다.
* 예시  
  MLNN을 이용해서 아래의 그림과 같이 16 \* 16 크기의 필기체를 인식하는 네트워크를 만든다고 하면, hidden layer가 한 층이고 100개의 뉴런을 가지고 있는 경우, 이 네트워크에 필요한 가중치(weight)와 편향(bias)는 총 28326개가 필요하게 된다.

만약, hidden layer를 한층 더 사용할 경우 파라미터 개수는 매우 많아지게 된다. 또한 아래의 그림 2와 같이 글자를 전체적으로 2픽셀씩 이동하게 되면 새로운 학습데이터로 처리해줘야 하는 문제점이 있을 뿐만 아니라, 글자의 크기, 회전, 변형이 있게 되면, 이에 대한 새로운 학습데이터를 넣어줘야 하는 문제가 있다.

글자의 형상은 고려하지 않고, raw data를 직접 처리하기 때문에 많은 양의 학습데이터가 필요하고, 따라서 학습시간이 길어지게 된다.

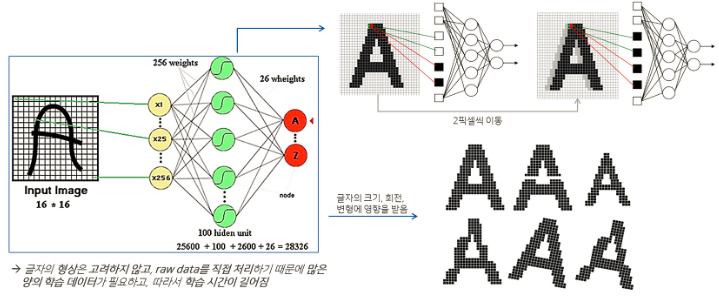
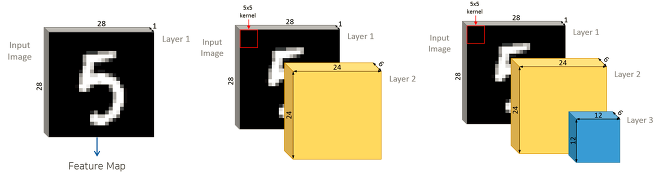


그림 2) MLNN의 문제점

##### 합성곱 계층(Convolutional Layer, Conv Layer)

* 이러한 MLNN의 문제를 해결하고자 만들어진 것이 합성곱 계층이다. 위의 [그림 2]의 필기체나 MNIST 데이터 같은 이미지 데이터는 일반적으로 채널, 가로, 세로 이렇게 3차원으로 구성된 데이터이다. Affine 계층에서는 이 3차원 데이터를 1차원 데이터(784=28\*28)로 바꿔 입력했지만 합성곱에서는 3차원 데이터(1, 28, 28)를 입력하고 3차원의 데이터로 출력하므로 형상을 유지할 수 있다. CNN에서는 이러한 입출력 데이터를 특징맵(Feautre Map)이라고한다.  
    
   그림 3) Conv Layer에서의 입출력 데이터 형상

##### 3-1) 합성곱 계층 – 연산

* 그럼, 합성곱 계층에서 연산이 어떻게 이루어지는지 알아보도록 하자. 데이터와 필터의 모양을 (높이, 너비)로 나타내고, 윈도우(Window)라고 부른다. 여기서 입력데이터는 (4, 4), 필터는 (3, 3)이고, 필터가 바로 Conv Layer의 가중치에 해당한다.

합성곱 연산은 필터의 윈도우를 일정한 간격으로 이동해가며 계산한다. [그림 4]를 통해 계산 방법을 알아보면. 합성곱 연산은 입력데이터와 필터간에 서로 대응하는 원소끼리 곱한 후 총합을 구하게 되며, 이것을 Fused Multiply-Add(FMA)라고한다.

마지막으로 편향은 필터를 적용한 후에 더해주게 된다.

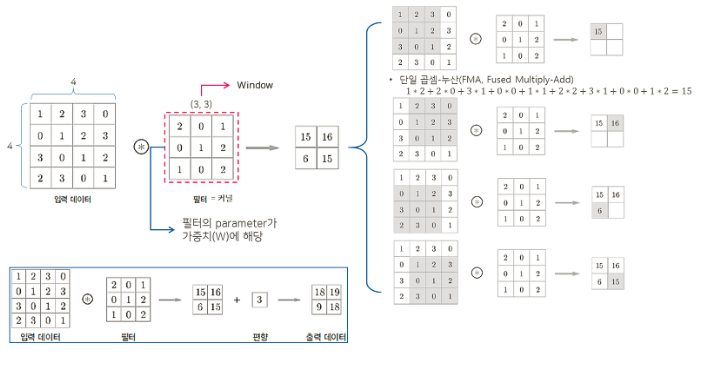


그림 4) 합성곱 연산 예제

##### 3-2) 합성곱 계층 – 패딩(padding)

* 패딩(Padding)은 합성곱 연산을 수행하기 전, 입력데이터 주변을 특정값으로 채워 늘리는 것을 말한다. 패딩(Padding)은 주로 출력데이터의 공간적(Spatial)크기를 조절하기 위해 사용한다. 패딩을 할때, 채울 값은 hyperparameter로 어떤 값을 채울지 결정할 수 있다. 주로 zero-padding을 사용한다.
* 패딩을 사용하는 이유는 패딩을 사용하지 않을 경우, 데이터의 Spatial 크기는 Conv Layer를 지날 때 마다 작아지게 되므로, 가장자리의 정보들이 사라지는 문제가 발생하기 때문에 패딩을 사용한다.

##### 3-3) 합성곱 계층 – 스트라이드(stride)

* 스트라이드는 입력데이터에 필터를 적용할 때 이동할 간격을 조절하는 것, 즉 필터가 이동할 간격을 말한다. 스트라이드 또한 출력 데이터의 크기를 조절하기 위해 사용한다. 스트라이드(Stride)는 보통 1과 같이 작은 값이 더 잘 작동하며, Stride가 1일 경우 입력 데이터의 spatial 크기는 pooling 계층에서만 조절하게 할 수 있다.

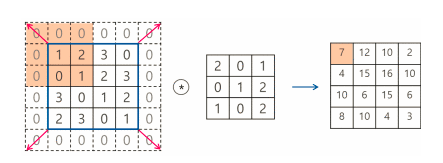
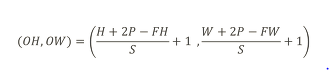
아래의 [그림 5]는 1폭 짜리 zero-padding과 Stride값을 1로 적용한 뒤 합성곱 연산을 수행하는 예제이다.  


그림 5) zero – padding, stride 적용한 합성곱 연산

##### 3-3) 합성곱 계층 – 출력크기 계산

* 패딩과 스트라이드를 적용하고, 입력데이터와 필터의 크기가 주어졌을 때 출력 데이터의 크기를 구하는 식은 아래 [식 1]과 같다.   
  

식 1) 출력 크기 계산식

* 아래의 [그림 6]은 패딩 1 스트라이드 1 일때의 출력데이터 크기를  구한 예제다.

출력크기가 정수가 아닌 경우에는 에러가 발생할 수 있는데, 보통 딥러닝 프레임워크에서는 반올림을 통해 에러 없이 작동한다.

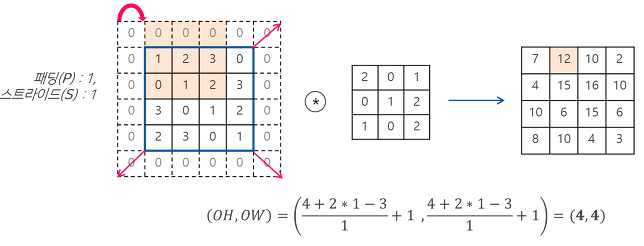


그림 6) 출력 크기 계산 예제

##### 풀링 계층(poolling layer)

* 풀링계층은 합성곱 계층의 패딩과 스트라이드처럼 데이터의 공간적 크기를 축소하는데 사용한다.
* 주로 합성곱 계층(Conv Layer)에서 출력데이터의 크기를 입력데이터의 크기 그대로 유지하고, 풀링계층(Pool 에서만 크기를 조절한다.
* 풀링에는 Max-Pooling과 Average pooling이 있는데 Max-Pooling은 해당영역에서 최대값을 찾는 방법이고, Average-Pooling은 해당영역의 평균값을 계산하는 방법이다. 이미지 인식 분야에서는 주로 Max-Pooling을 사용한다.
* 풀링계층에서는 풀링의 윈도우 크기와 스트라이드 값은 같은 값으로 설정한다. 아래의 [그림 7]에서는 풀링의 윈도우 사이즈는 (2, 2)이며 스트라이드는 2로 설정하였다.
* [그림 7]은 풀링 종류 중 맥스풀링을 사용하여 처리한 예제다.

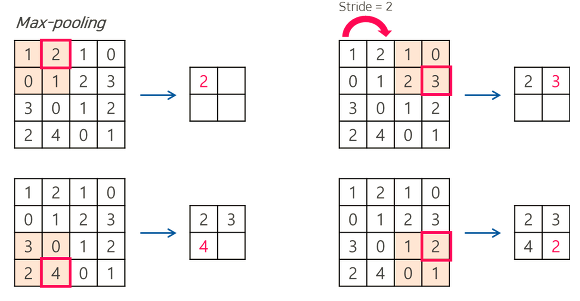


그림 7) Max-pooling 예제

# 참고문헌

##### <https://github.com/youbeebee/deeplearning_from_scratch/tree/master/ch7.CNN>

##### 합성곱 신경망(CNN, Convolutional Neural Network) <http://excelsior-cjh.tistory.com/79>