CNN(Convolutional Neural Network)

1. August 2018

CNN이 무엇이고, 어떻게 작동하는지 살펴보자.

요즘 Youtube를 통해 김성훈교수님의 '머신러닝과 딥러닝 BASIC' 강의를 듣고 있습니다. CNN 전까지의 내용은 대체로 머신러닝의 기초를 다루고 있기 때문에 크게 어려운 점이 없어서 따로 리뷰할 생각이 없었습니다. CNN부터는 내용을 정리하는 겸 리뷰를 하기로 했고, 앞으로는 관련한 논문들에 대한 재구현을 해보고 리뷰를 할 생각입니다.

시작하기에 앞서 저의 딥러닝 공부에 막대한 도움을 주고 있는 령우와 훈석이에게 감사의 말을 전합니다.

CNN(Convolutional Neural Network)는 무엇일까요? 수학하는 사람들에게는 convolution이라는 단어가 익숙하지만 전공이 다른 분들에게는 이 단어가 익숙하지 않을 수도 있을 것 같습니다. 하지만 크게 걱정하지 않아도 되는 것이, 제가 수학을 공부하며 보았던 convolution과는 다소 거리가 있었기 때문에, 큰 부담없이 새로운 개념으로써 이해해도 전혀 문제가 없습니다.

CNN은 말 그대로 'convolution'이라는 작업이 들어가는 NN을 의미합니다. 그럼 **convolution**이 무엇인지 알아야

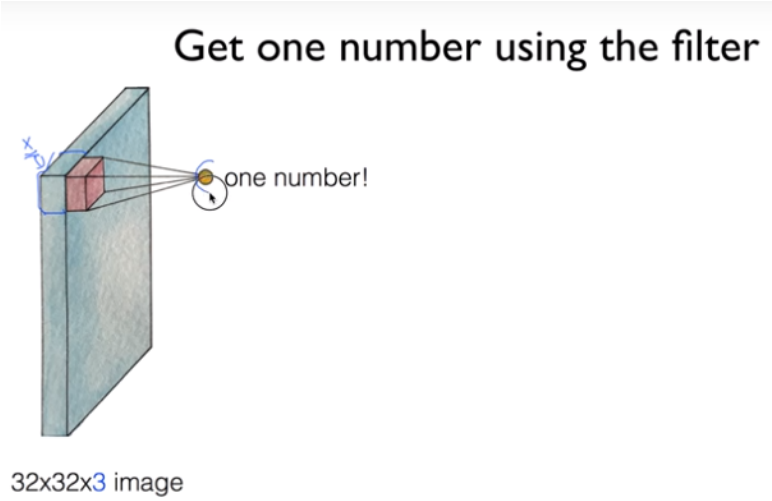
Search …

RECENT POSTS

- Python & Vectorization
- Deep Learning resources
- Deep Learning and Art
- Simultaneous hyperparameter tuning on multiple prediction methods in a limited resources…
- How to Turn Your Business into a Cognitive Enterprise with AI Technologies?

할텐데요. 이 과정을 예를 들면서 묘사해보고자 합니다.

초창기에 CNN을 개발한 분들이 고양이가 보는 것마다 자극 받는 뇌의 위치가 다른 것을 보고 아이디어를 얻어 CNN을 만들었다고 합니다. *즉, image 전체를 보는 것이 아니라 부분을 보는 것이 핵심 아이디어입니다.* 이 '부분'에 해당하는 것을 **filter**라고 합니다.

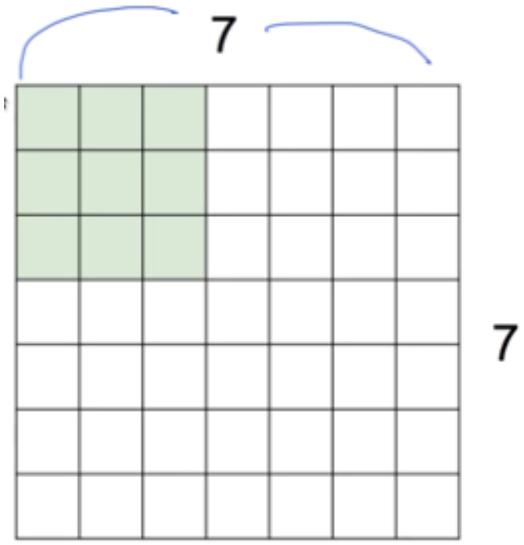


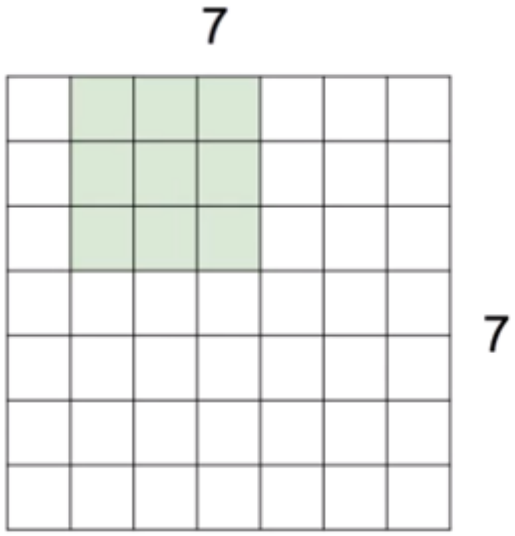
빨간색에 해당하는 것이 filter입니다. (출처: 강연 동영상)

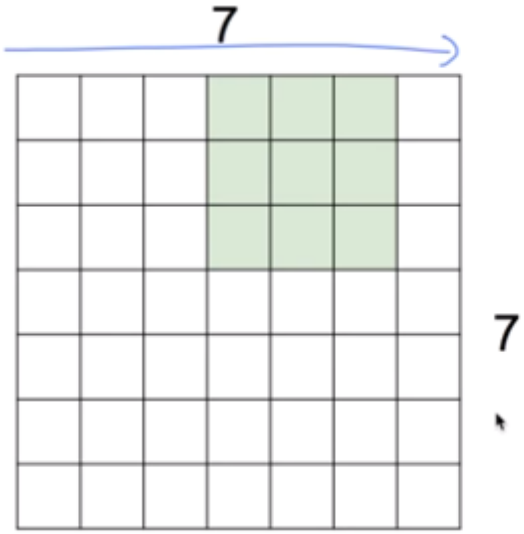
filter와 관련하여 좀 더 자세히 알아보기 위해 편의상 image의 사이즈를 7×7이라고 가정하겠습니다. 그러면 우리는 여기서 filter의 size를 3×3이라고 (임의로) 정할 수 있습니다.

다음으로 우리는 위 사진과 같이 각 부분부분마다 하나의 숫자를 뽑아내고 싶습니다. (뽑아낸다고 썼지만 ***pooling****이라는 뜻은 아닙니다.* pooling에 관하여는 다른 포스트를 통해 설명하겠습니다.) 현재 상황에서 보자면, filter의 size가 3×3이므로 총 9개의 parameter가 있고, 이를 x라고 하면 weight W에 대하여 Wx+b나 ReLU(Wx+b)와 같은 activation을 거쳐 하나의 실수로 출력할 수 있습니다.

우리의 image는 7×7이니까 3×3의 filter를 한 번 적용해서는 전체를 볼 수 없겠죠? 즉, filter를 '옮겨가며' 위와 같은 작업을 반복할텐데, 아래나 옆으로 옮긴다면, '얼마나' 옮겨야 하는가? 하는 자연스러운 질문을 할 수 있게 됩니다. 이 때의 옮기는 양을 우리는 **stride**라고 부릅니다. 우리의 경우에서 stride를 1로 준다면, 아래 그림과 같이 전체 image를 훑을 것입니다. (이 과정에서 쓰이는 weight W는 같습니다.)







각 스텝마다 1개의 value를 얻기 때문에 이와 같은 과정을 반복하면 총 5×5의 output을 얻게 됩니다. 잘 보면 아시겠지만 꽤나 규칙적인데요. 실제로 output의 size는 아래와 같은 공식으로 얻을 수 있겠습니다.

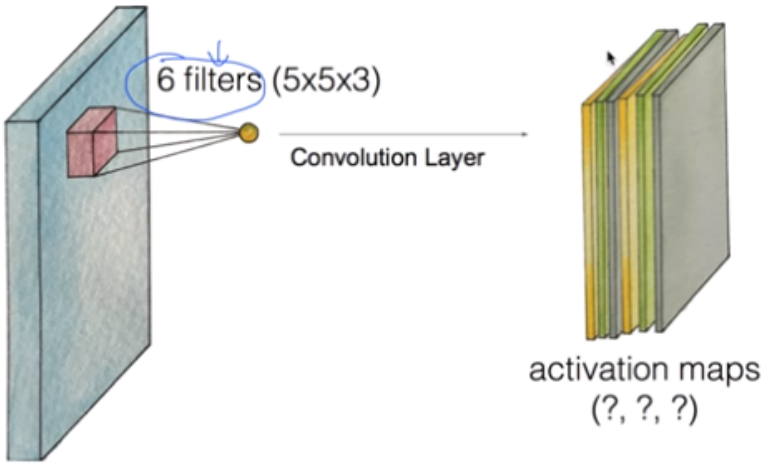
> *Total size ; N (i.e. N x N)*

> *Filter size ; F (i.e. F x F)*

> *Output size ; k:=(N-F)/stride + 1 where k is an integer.*

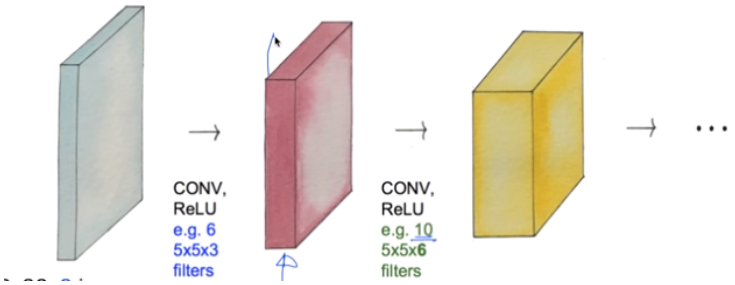
현재의 예에서 k = (7−3)/1 + 1 = 5임을 알 수 있습니다. 위와 같이 계산해서 나온 k가 정수가 아닌 경우(e.g. N=7, F=3, stride=3)는 제외하도록 합니다. 이렇게 나온 output을 **convolution layer**라고 부릅니다.

지금까지 한 것은 결국 filter가 하나당 activation이 하나 주어지면 이에 따른 convolution layer를 얻을 수 있다는 것이었습니다. 즉, filter가 달라지면 이에 따른 activation을 다르게 줄 수 있으니까(꼭 달라야 하는 것은 아니지만), 이로부터 나오는 새로운 convolution layer를 생각할 수 있을 것입니다. 예를 들어 6개의 filter로부터 6개의 convolution layer를 얻었다고 합시다. 현재 우리의 예에서는 N=7, F=3 이기 때문에 5×5짜리 convolution layer를 6개 얻은 셈입니다. 이를 하나로 합치면 5x5x6짜리 convolution layers(혹은 **activation maps**라 부릅니다.)를 얻게 됩니다. 이렇게 activation maps까지 얻는 일련의 과정을 **convolution**이라고 합니다. (상황에 따라서는 filter에서 하나의 값을 얻는 것 자체를 convolution이라고 하기도 한다고 합니다. 어쨌든 초점은 기존의 것으로부터 filter를 거쳐 새로운 것을 만들어내는 것을 convolution으로 이해하면 일맥상통합니다.)



사진에서의 5×5 부분을 우리의 경우에선 3×3으로 이해하면 됩니다.

이러한 convolution은 한 번만 할 수도 있지만, 상황에 따라서는 여러번의 convolution을 해도 될 것입니다. 우리의 예에서는 activation maps가 5x5x6이 나왔으므로, 이를 새로운 image로 보고 앞에서의 과정을 반복하면 되겠습니다. (각 convolution마다 쓰이는 weight은 다른 딥러닝 기법들과 마찬가지로 학습을 통해 얻을 수 있습니다.)



이렇게 여러번 해도 해도 됩니다.

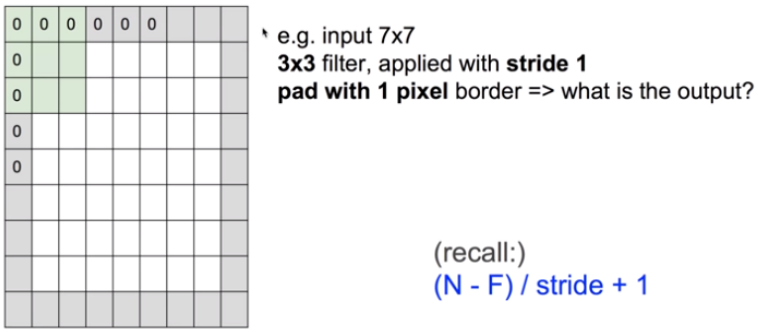
추가적으로 **padding**에 대한 이야기를 해보고자 합니다. 위에서 convolution하는 과정을 잘 보면 convolution layer의 사이즈 k는 일반적으로 N보다 작거나 같습니다. (stride≥1, F≥1이기 때문이죠.)

> $k = (N-F)/stride + 1 \leq N-F+1 = N-(F-1) \leq N$

우리의 경우에서는 7×7짜리 data가 convolution을 거쳐 5×5의 data가 됩니다. 즉, 상당량의 *데이터 손실*을 일으키게 됩니다. 여기서 말하는 데이터의 손실이란 수학적으로는 차원이 낮아지는 것을 의미합니다. (수학에 큰 거부감이 있는 분들은 바로 아래 단락은 무시하고 다음 단락으로 진행하셔도 됩니다.)

> *RGB의 양을 무시하고(어차피 곱해지는 양은 같으니까) 7×7과 5×5만 살펴보면, 49차원으로 표현되는 data가 25차원으로 확 떨어지는 것이죠. 일반적으로 convolution을 거치기 전부터 25차원에 data가 embedding 되어있지 않기 때문에 데이터 손실은 불가피 할 것입니다.*

결국 우리의 목표는 7×7의 형태를 그대로 유지하는 것입니다. Convolution을 거치고도 유지가 되려면 어떻게 해야할까요? 강연 동영상에서는 **zero padding**이라는 technique을 사용하고 있습니다. 아래 그림과 같이 7×7 size의 image를 위아래 양옆에 0을 덧댐으로써 9×9로 확장합니다.



이렇게 확장된 9×9 image에 대해 위에서 했던 convolution을 하면 7×7짜리 convolution layer를 얻게 됩니다! 원했던 것과 같이 같은 size의 convolution layer를 얻어 데이터의 손실을 막게 되었습니다.

여기서 가장 자연스러운 질문은 '얼마나' padding 해야 하는데? 라는 질문일 것입니다. 위에서는 padding한 양이 1개죠. 일반적으로는 다음과 같은 공식을 통해 알 수 있습니다.

> *Total size ; N*

> *Filter size ; F*

> *Stride ; s*

> *Padding size ; p*

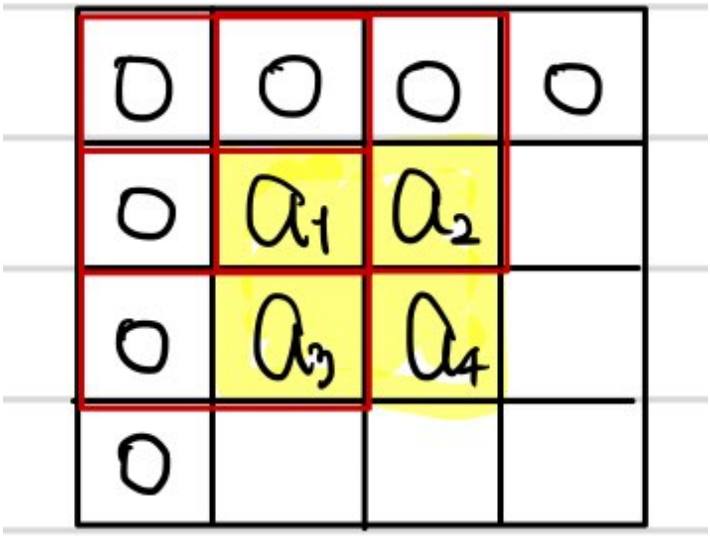
> *Convolution layer size after padding; k = [(N+2p)-F]/s + 1*

> *k=N if and only if p=[(N-1)s+F-N]/2*

위의 공식을 적용해보면, s=1일 때 F=3이면 p=(3-1)/2=1이므로 공식이 잘 적용되는 것을 확인할 수 있습니다.

Padding, 물론 손실을 막아주는 것은 굉장히 좋지만, 이에 따라 필요없는 부분이 생기는 것도 고려해야 합니다. 원래 데이터에는 0을 붙인 만큼이 없었기 때문에, 우리가 원래 필요한 7×7의 데이터 말고도 noise가 발생하게 됩니다. 그럼에도 convolution without padding으로 얻는 데이터 손실보다는 더 나으니까 padding을 쓰겠죠? 우리는 이런 것을 **trade-off**라고 합니다.

말로만 하면 이해가 잘 안되니까, 제가 그린 아래의 그림으로 좀 더 추가설명을 해보겠습니다.



실제로 계산하고 싶은 노란색 영역 말고도 빨간색 영역의 값이 noise로 생긴다.

Filter size를 2라고 하면 위와 같은 형태로 훑으며 값을 얻을 것입니다. 왼쪽 위만 생각해 보았을 때, padding하기 전에는 노란색 영역의 값만 필요한 것이었는데, padding을 한 후에는 빨간색 영역 만큼이 추가로 noise로 발생하게 됩니

다. 이 때 noise가 주는 오류를 최소화하기 위해 0을 padding 했다고 생각합니다.

생각보다 이번 포스트가 너무 길어진 관계로 **pooling**에 대해서는 다음 포스트에서 이어쓰도록 하겠습니다.

Source: Deep Learning on Medium

"ㄷㅍㅅ" - 딥러닝 프로젝트 스쿨 @ 마이캠퍼스
Source: Deep Learning on Medium 마이캠퍼스Dec 28딥러닝/머신러닝 분야 취업 및
28. December 2018
Similar post

Udacity Deep Learning Nanodegree 수료 후기
Source: Deep Learning on Medium Amy SonDec 21상업적 홍보와 관련없는, 글
22. December 2018
Similar post

머신러닝부터
Python을 이용한 머신러닝, 딥러닝 실전 개발 입문
2018.04.02.월지금까지는 데이터를 수
2. April 2018
Similar post

« VERY QUICK SETUP OF CAFFENET (ALEXNET) FOR IMAGE CLASSIFICATION USING NVIDIA-DOCKER 2.0

PYSPARK INTEGRANDO MLFLOW (TRACKING MODELS) »