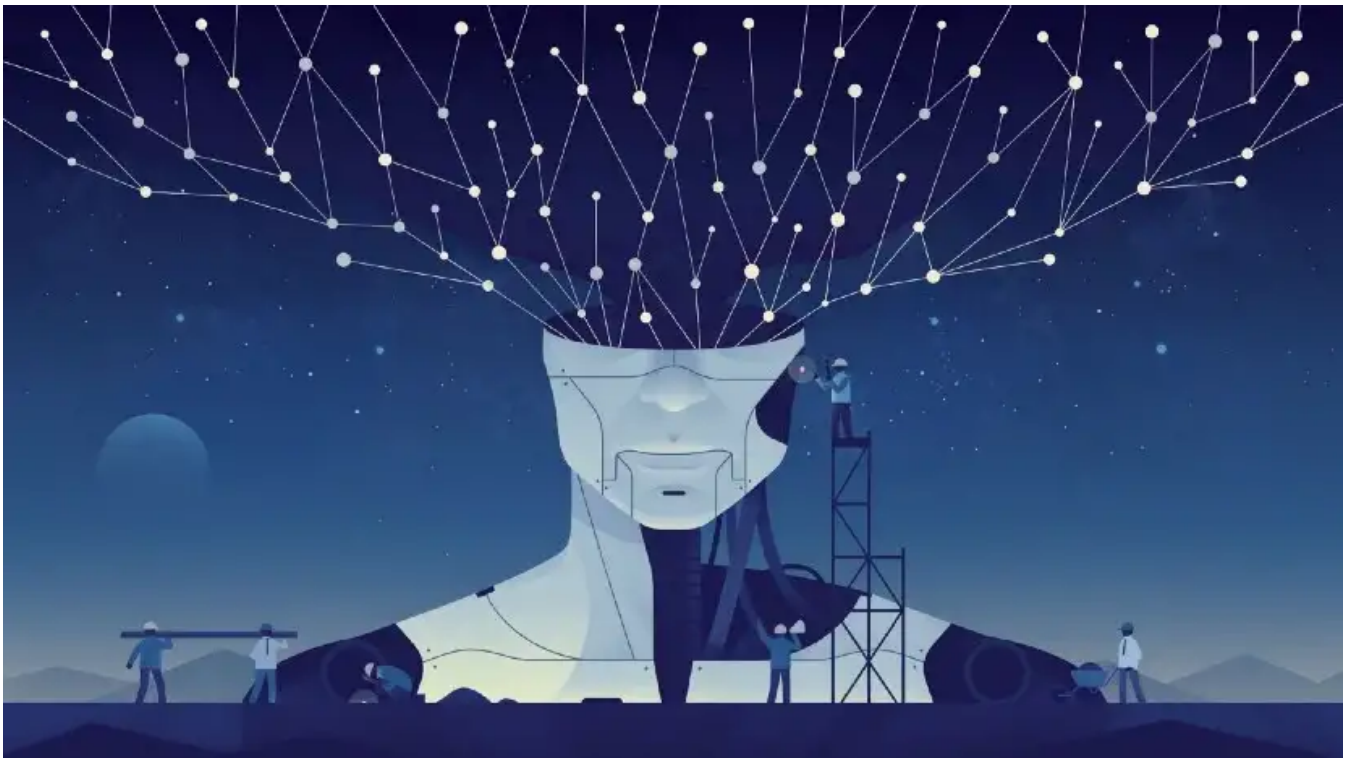
Irene Ban Follow

Oct 27, 2020 · 16 min read

[딥러닝/머신러닝] CNN(Convolutional Neural Networks) 쉽게 이해하기



필자는 뉴욕에서 머신러닝 전공으로 석사학위 취득하였습니다. 학교 수업을 듣고 실습 과제를 하며 항상 느꼈던 점은 이 분야는 무엇보다도 튼튼한 이론지식이 중요하다는 것입니다. 물론 어느 정도의 코딩실력도 밑받침이 되어야 하지만 경험상 모델에 대한 정확한 이해가 없으면 좋은 코드를 짜내기 쉽지 않았습니다. 때문에 학교에서 배운 이론 개념을 바탕으로 기록의 의미에서 머신러닝 관련 포스팅을 시작합니다. 전공자뿐만 아니라 배경지식이 없는 사람이여도 쉽게 이해할 수 있도록 노력하여 썼습니다. 중간중간 더 나아가 심화수준의 개념이해를 위해선 관련 링크를 걸어두었습니다.

> 비루한 석사생일 뿐입니다...잘못된 정보(혹은 오타)를 발견하셨을 경우 꼬옥 알려주세요.

이번 포스팅은 **CNN** 모델에 관한 개념 설명입니다. 이 포스팅의 목차는 다음과 같습니다.

1. CNN이 무엇일까?
2. CNN의 주요 컨셉들
3. CNN의 전체적인 네트워크 구조
4. 매개변수(parameter)와 Hyper-매개변수(hyper-parameter)

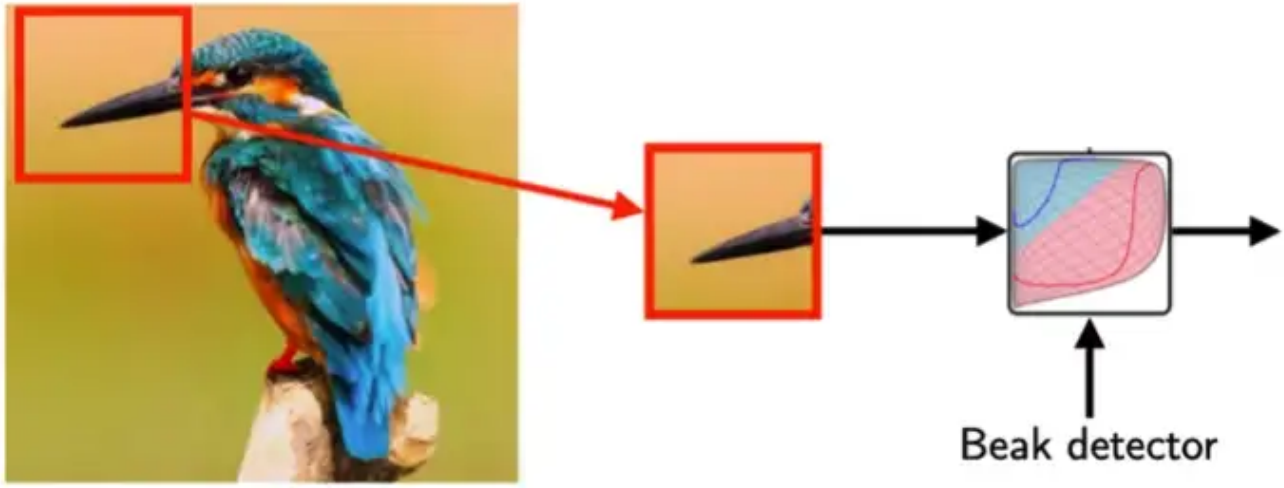
1. CNN이 무엇일까? — 큰 그림 그리기

CNN은 **Convolutional Neural Networks**의 약자로 딥러닝에서 주로 이미지나 영상 데이터를 처리할 때 쓰이며 이름에서 알수있다시피 Convolution이라는 전처리 작업이 들어가는 Neural Network 모델입니다.

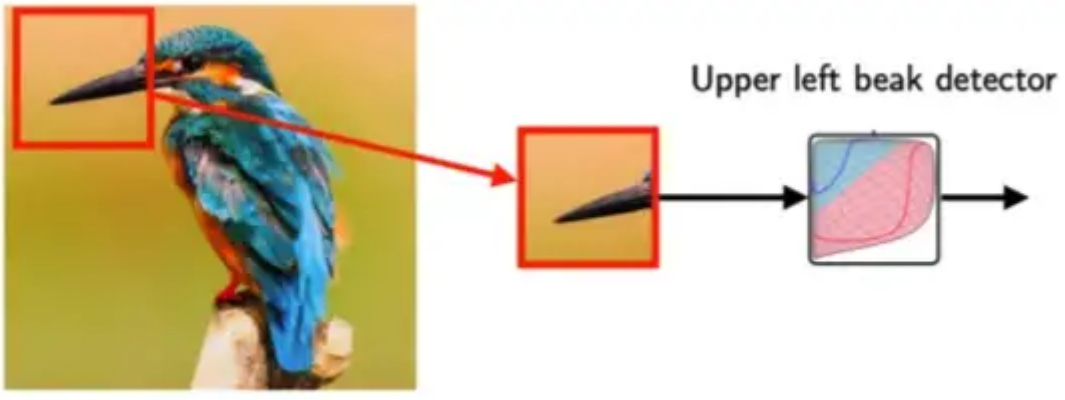
그렇다면 왜 CNN이라는 방법을 쓰기 시작했을까요? 이에 대한 답은 일반 DNN(Deep Neural Network)의 문제점에서부터 출발합니다. 일반 DNN은 기본적으로 1차원 형태의 데이터를 사용합니다. 때문에 (예를들면 1028x1028같은 2차원 형태의)이미지가 입력값이 되는 경우, 이것을 flatten시켜서 한줄 데이터로 만들어야 하는데 이 과정에서 이미지의 공간적/지역적 정보(spatial/topological information)가 손실되게 됩니다. 또한 추상화과정 없이 바로 연산과정으로 넘어가 버리기 때문에 학습시간과 능률의 효율성이 저하됩니다.

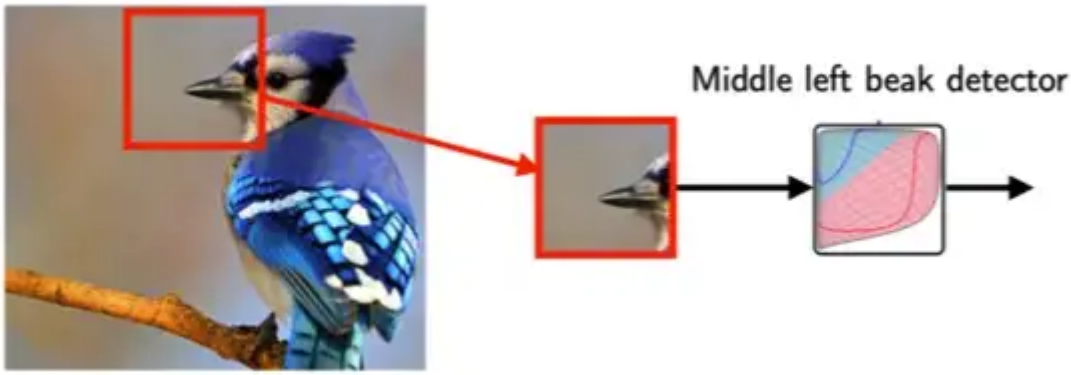
이러한 문제점에서부터 고안한 해결책이 CNN입니다. CNN은 이미지를 날것(raw input) 그대로 받음으로써 공간적/지역적 정보를 유지한 채 특성(feature)들의 계층을 빌드업합니다. CNN의 중요 포인트는 이미지 전체보다는 부분을 보는 것, 그리고 이미지의 한 픽셀과 주변 픽셀들의 연관성을 살리는 것입니다.

왜 CNN을 쓰는지 조금 더 쉽게 예를 들어 봅시다.



예를 들어 우리는 어떠한 이미지가 주어졌을때 이것이 새의 이미지인지 아닌지 결정할 수 있는 모델을 만들고 싶다고 가정합시다. 그렇다면 새의 주요 특징인 새의 부리가 중요한 포인트가 될 수 있습니다. 때문에 모델이 주어진 이미지에 새의 부리가 있는지 없는지 판가름 하는것이 중요 척도가 될 것입니다. 하지만 새의 전체 이미지에서 새의 부리 부분은 비교적 작은 부분입니다. 때문에 모델이 전체 이미지를 보는 것 보다는 새의 부리 부분을 잘라 보는게 더 효율적이겠죠? 그것을 해주는것이 CNN입니다. CNN의 뉴런이 패턴(이 경우에서는 새의 부리)을 파악하기 위해서 전체 이미지를 모두 다 볼 필요가 없습니다.





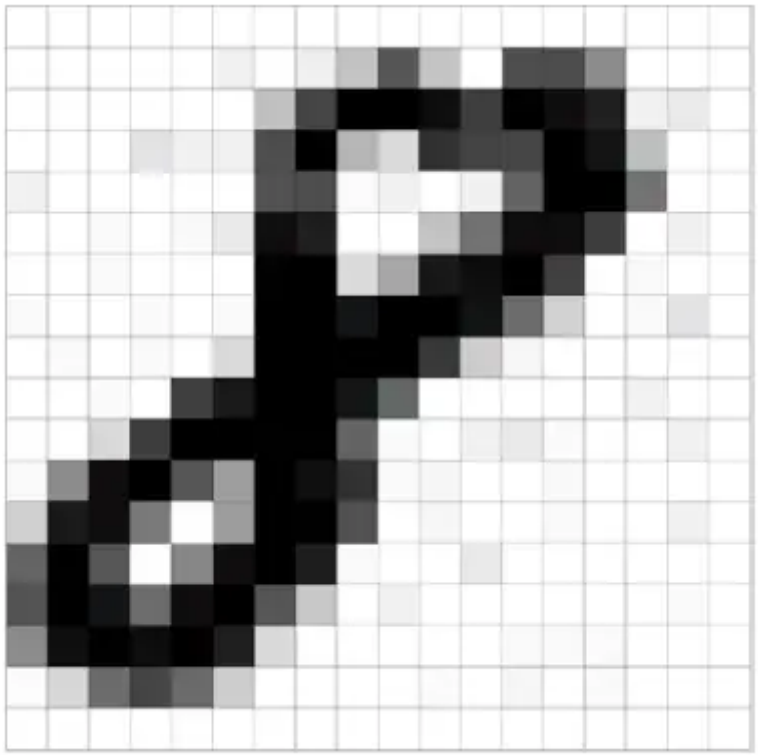
또한 위의 두 이미지를 보면 새의 부리부분이 이미지의 다른 부분에 위치해 있다는 것을 알 수 있습니다; 위의 이미지는 이미지의 새의 부리가 왼쪽 상단에 위치한 반면 아래의 이미지는 상단 가운데 부분에서 약간 왼쪽 부분에 위치. 때문에 전체적인 이미지보다는 이미지의 부분 부분을 캐치하는 것이 중요하고 효율적일 수 있습니다.

2. CNN의 주요 컨셉들

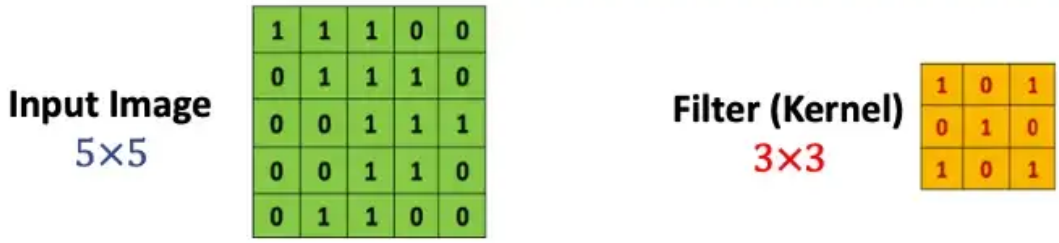
그렇다면 이제 대충 CNN의 큰 그림을 그려보았으니 주요 컨셉들을 알아봅시다.

2.1. Convolution의 작동 원리

우선 2차원의 이미지를 예로 들어 봅시다. 2차원 이미지는 픽셀 단위로 구성되어 있습니다.



위의 이미지는 MNIST Dataset에서 추출한 샘플중 하나이며 손글씨로 쓰여진 '8'의 gray scale 이미지 입니다. 자세히 보면 28x28단위의 픽셀로 구성되어 있습니다. 우리는 이 데이터 입력값을 28x28 matrix(행렬)로 표현할 수 있습니다. 즉, 우리는 2차원 이미지를 matrix로 표현할 수 있다는 뜻입니다. 우리가 CNN에 넣어줄 입력값은 이렇게 *matrix*로 표현된 이미지 입니다.



조금 더 간략한 예를 위해 위와 같이 5x5의 matrix로 표현된 이미지 입력값이 있다고 가정합시다. 그리고 CNN에는 필터(커널)가 존재합니다. 위의 예시의 경우에는 3x3크기의 필터입니다. 쉽게 표현하면 이 하나의 필터를 이미지 입력값에 전체적으로 훑어준다고 생각하면 됩니다. 즉 우리의 입력값 이미지의 모든 영역에 같은 필터를 반복적용해 패턴을 찾아 처리하는 것이 목적입니다. 그렇다면 이미지 위에 이 필터를 훑어줄때 어떤 일이 일어날까요? 정확히 말하면 이 필터를 이용해 연산처리를 해주는 것입니다. 그리고 이 연산처리는 matrix와 matrix간의 *Inner Product*라는 것을 사용합니다.

여기서 잠깐, *Inner product*이란 무엇일까요? 이것은 이산수학의 아주 기초이므로 간단히 이미지를 보고 넘어가겠습니다.

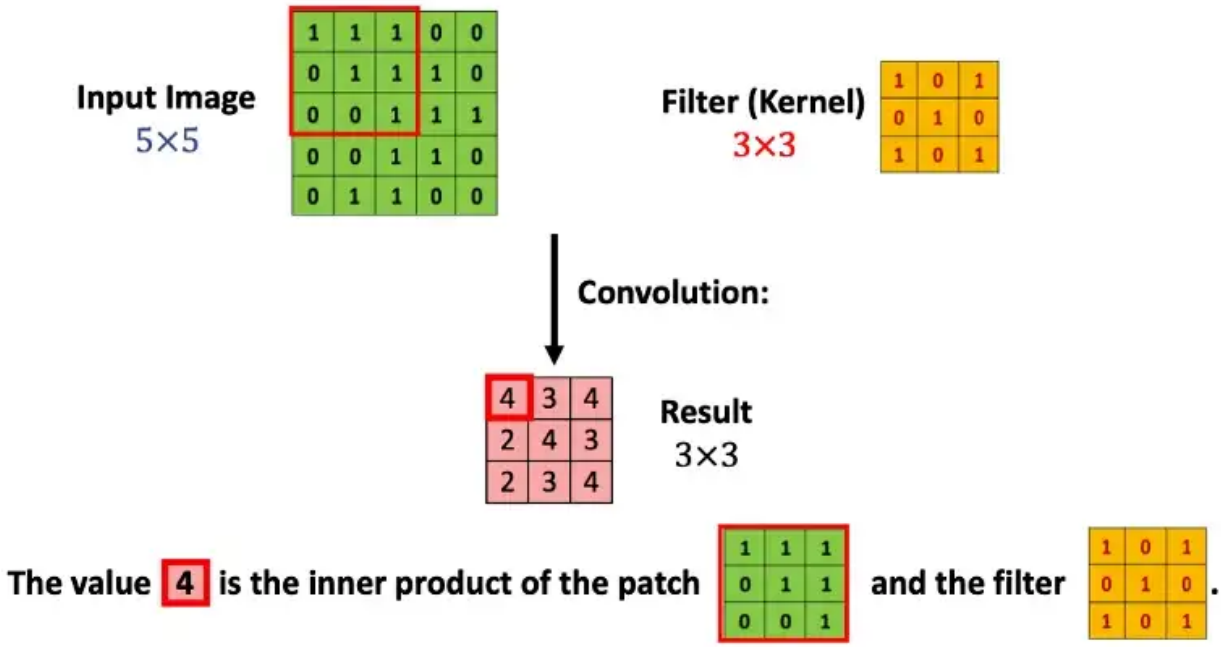
- $\mathbf{A} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}$.

- Inner product:

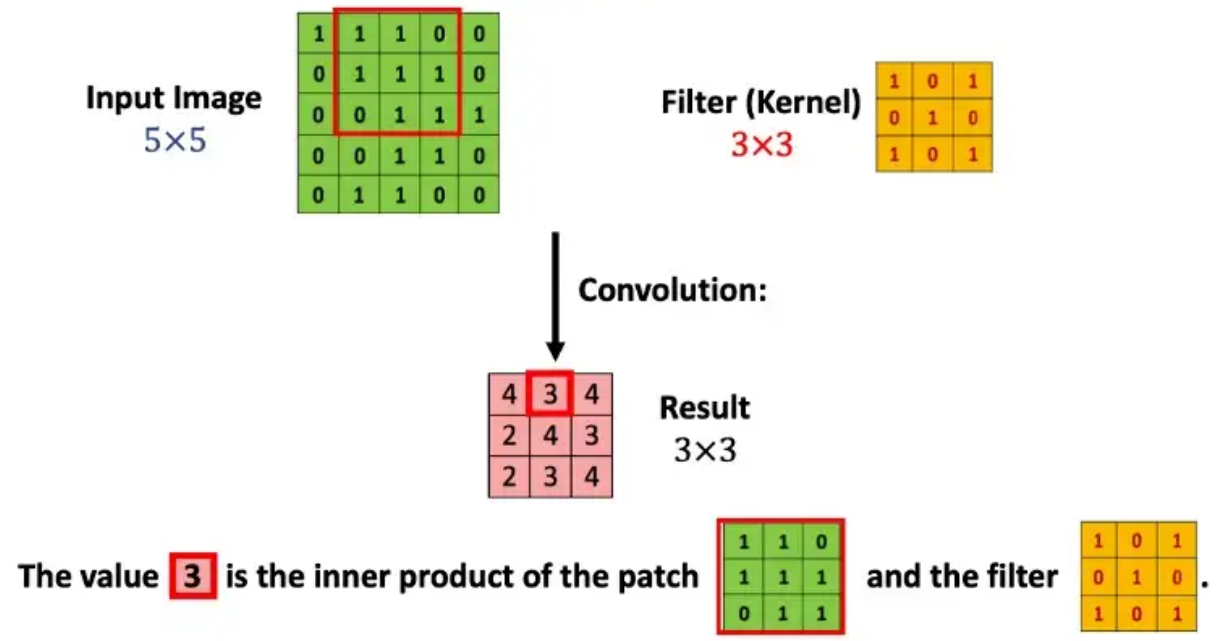
$$\langle \mathbf{A}, \mathbf{B} \rangle = \sum_i \sum_j a_{ij} b_{ij} = 70.$$

쉽게 말해 같은 크기의 두 matrix를 놓고 각 위치에 있는 숫자를 곱해 모두 더해주는 것입니다.

다시 본론으로 되돌아와서 어떻게 이 이미지 입력값 matrix의 각 부분과 필터가 연산처리 되는지 그림으로 살펴보겠습니다.



우선 빨간 테두리 부분의 matrix와 필터의 Inner product연산을 해주면 그 결과값은 4가 됩니다. 그리고 필터를 한칸 옆으로 옮겨주면,



이번엔 이 빨간 테두리 부분의 matrix와 필터의 연산처리를 해주어 결과값 3을 얻어냅니다. 이러한 과정을 전체 이미지 matrix에 필터 matrix를 조금씩 움직이며 반복해 줍니다. 그렇게 반복하여 얻어낸 결과값이 분홍색으로 표시된 matrix입니다.

자세히 보면 결과값의 크기(단어 선택의 혼동이 있을 수 있습니다. 여기서 크기란 matrix의 dimension을 가르킵니다)는 3x3이라는 것을 파악할 수 있습니다. 왜 그런지 다들 눈치 채셨나요? 5x5로 구성된 칸들에 3x3크기의 블록을 움직인다고 생각해보면

너비와 높이 둘다 3번씩 놓을 수 있기 때문입니다. 이것을 수학적으로 표현해보면 다음과 같습니다.

```
입력값: d_1 x d_2
필터: k_1 x k_2
결과값: (d_1 - k_1 + 1) x (d_2 - k_2 + 1)
```

2.2 Zero Padding

방금 우리가 살펴본 Convolution처리를 보면 5x5크기의 이미지에 필터처리를 해주었더니 결과값의 크기가 3x3로 줄어들었다는 것을 알 수 있습니다. 즉 손실되는 부분이 발생한다는 뜻입니다. 이런 문제점(?)을 해결하기 위해 Padding이라는 방법을 사용할 수 있습니다. 쉽게 말해 0로 구성된 테두리를 이미지 가장자리에 감싸 준다고 생각하면 됩니다.

그렇다면 위의 경우 5x5크기의 이미지 입력값이 7x7크기 `(d_1 + 2) x (d_2 + 2)` 가 될 것입니다. 이 상태에서 3x3필터를 적용해줄 경우 똑같이 5x5크기의 결과값을 얻을수 있겠죠? 이렇게 되면 입력값의 크기와 결과값의 크기가 같아졌음, 즉 손실이 없어졌음을 볼 수 있습니다.

2.3 Stride

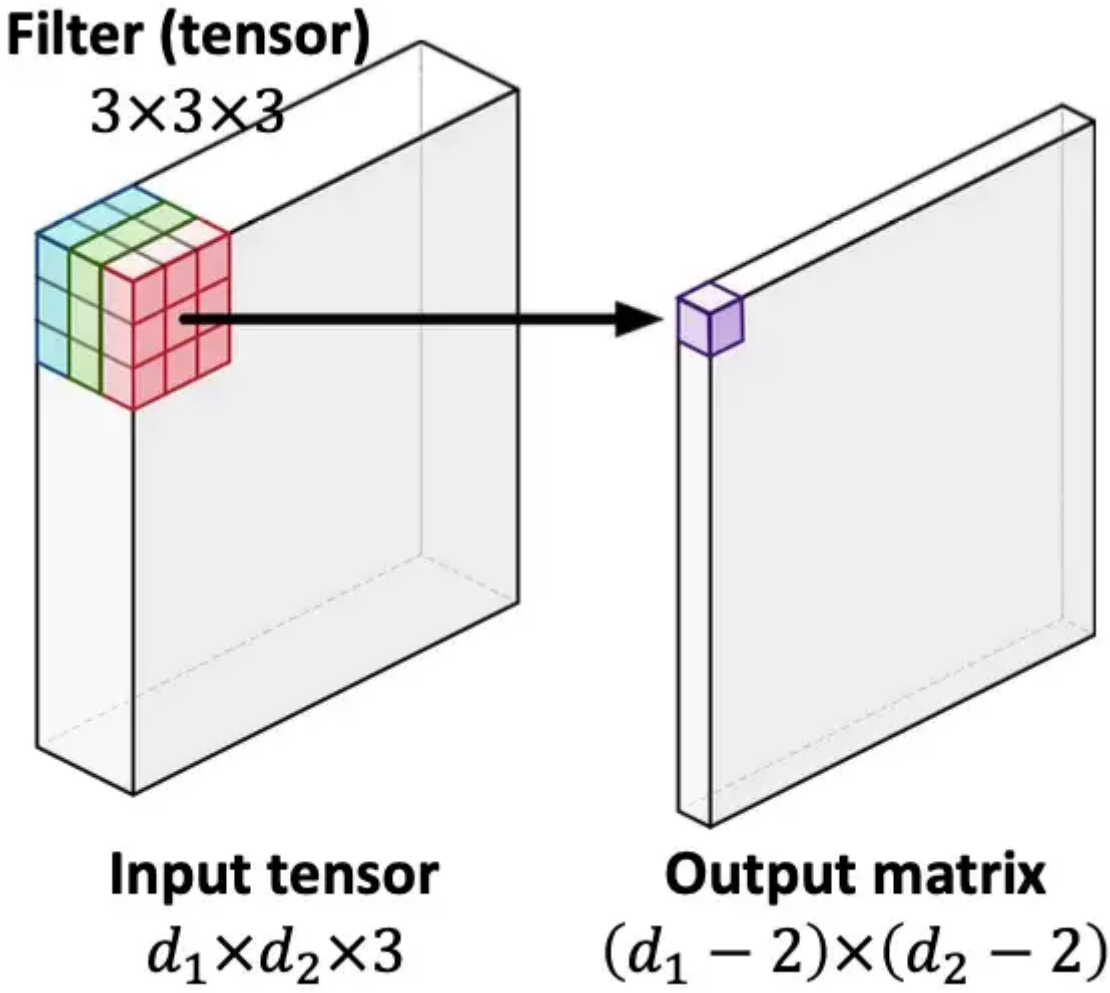
Stride는 쉽게 말해 필터를 얼마만큼 움직여 주는가에 대한 것입니다. stride값이 1일경우 필터를 한칸씩 움직여 준다고 생각하면 됩니다(즉, 위의 예시의 경우 stride값이 1일 때 입니다). 참고로 stride-1이 기본값이고 stride는 1보다 큰 값이 될 수도 있습니다. stride값이 커질 경우 필터가 이미지를 건너뛰는 칸이 커짐을 의미하므로 결과값 이미지의 크기는 작아짐을 의미합니다. stride값을 적용한 수학적 표현은 다음과 같습니다.

```
입력값: d_1 x d_2
필터: k_1 x k_2
stride: 1
결과값: (((d_1 - k_1) / s) + 1) x (((d_2 - k_2) / 2) + 1)
```

2.4 The Order-3 Tensor

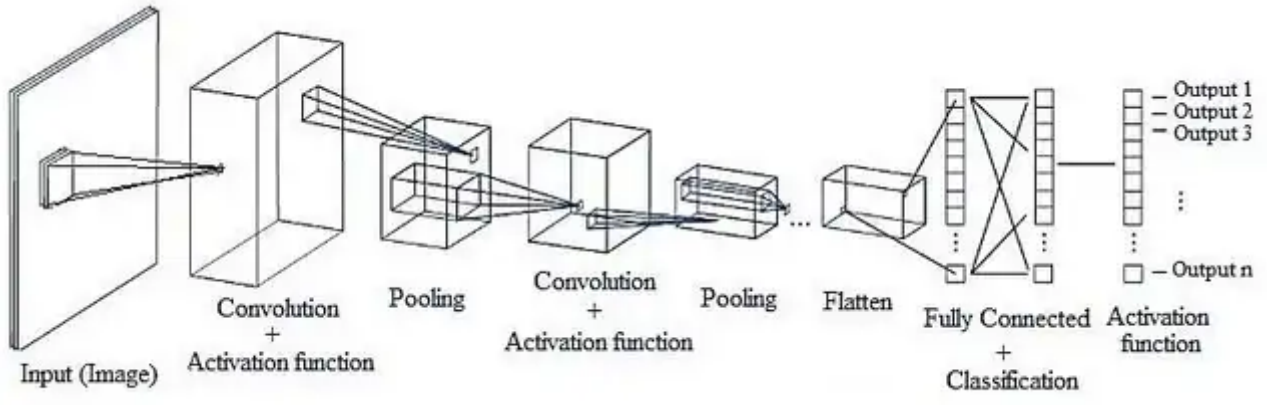
지금까지 우리는 2차원 이미지의 입력값을 살펴보았습니다. 물론 입력값이 3차원인 경우도 존재합니다. 예를들면 컬러이미지는 R, G, B의 세가지 채널로 구성되어 있기 때문에 `d_1 x d_2 x 3` 과 같은 삼차원의 크기를 갖습니다. 이러한 모양을 order-3 텐서

라고 칭합니다. 유사하게 익숙한 2차원의 이미지는 order-2 텐서(즉 matrix)라고도 칭합니다.



위의 이미지와 같이 이 경우에 필터는 k1xk2x3의 크기를 가진 order-3 텐서가 됩니다. 연산처리는 똑같이 Inner product를 사용합니다. 그러면 당연히 결과값의 모양은 matrix가 되겠죠?

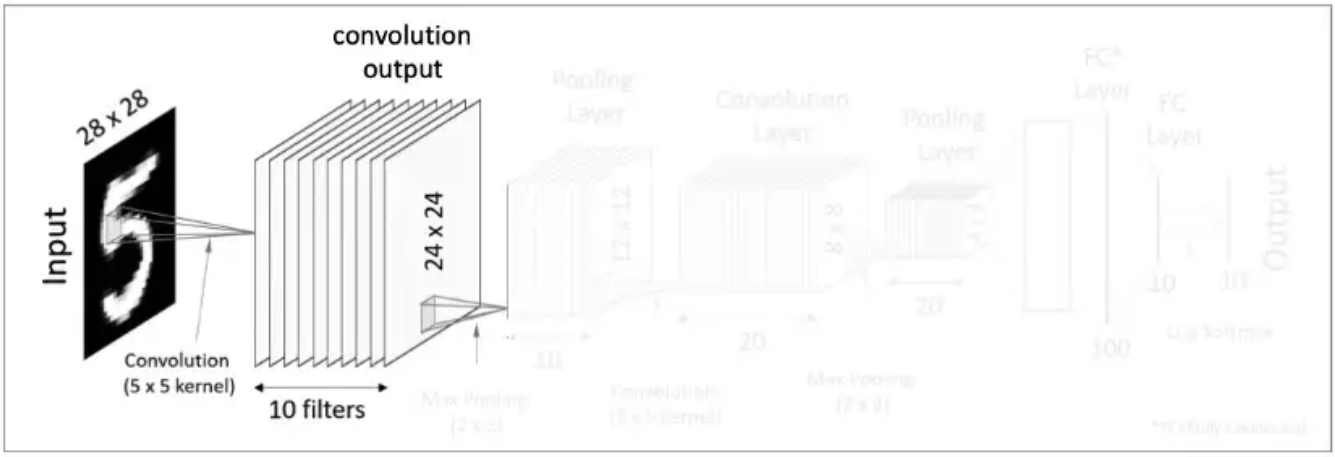
3. CNN의 전체적인 네트워크 구조



CNN의 구조는 기존의 완전연결계층(Fully-Connected Layer)과는 다르게 구성되어 있습니다. 완전연결계층(또는 Dense Layer이라고도 합니다)에서는 이전 계층의 모든 뉴런과 결합되어있는 Affine계층으로 구현했지만, CNN는 *Convolutional Layer*과 *Pooling Layer*들을 활성화 함수 앞뒤에 배치하여 만들어집니다.

이렇게 설명하면 너무 추상적이니 순서대로 각 계층을 자세히 살펴보겠습니다.

3.1. 첫번째 Convolutional Layer



우선 우리에게 주어진 입력값은 28x28크기를 가진 이미지입니다. 이 이미지를 대상으로 여러개의 필터(커널)을 사용하여 결과값(feature mapping이라고도 합니다)을 얻습니다. 즉 한개의 28x28 이미지 입력값에 10개의 5x5필터를 사용하여 10의 24x24 matrics, 즉 convolution 결과값을 만들어 냈습니다. 그 후 이렇게 도출해낸 결과값에 Activation function(예를 들면 ReLU function)을 적용합니다. 이렇게 첫번째 Convolutional Layer이 완성되었습니다. 우리는 여기서 한 Convolutional Layer는 Convolution처리와 Activation function으로 구성되어 있다는 것을 알 수 있습니다.

```
A Convolutional Layer = convolution + activation
```

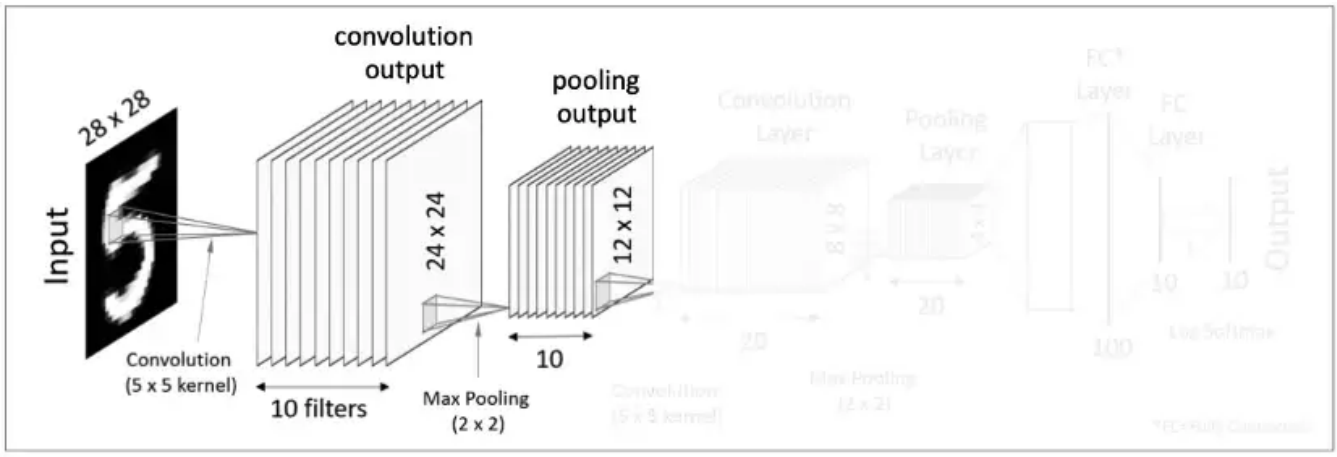
> 잠깐, 왜 활성함수*(Activation function)*을 쓰고 이건 무엇일까*?*

간단히 말하자면 선형함수(linear function)인 convolution에 비선형성(nonlinearity)를 추가하기 위해 사용하는 것입니다. 활성함수에 대한건 차후의 포스팅에서 자세히 다뤄보도록 하고 우선은 제 기준에서 잘 설명해 놓았다고 생각한 포스팅 링크들을 걸어두겠습니다.

- https://medium.com/@snaily16/what-why-and-which-activation-functions-b2bf748c0441

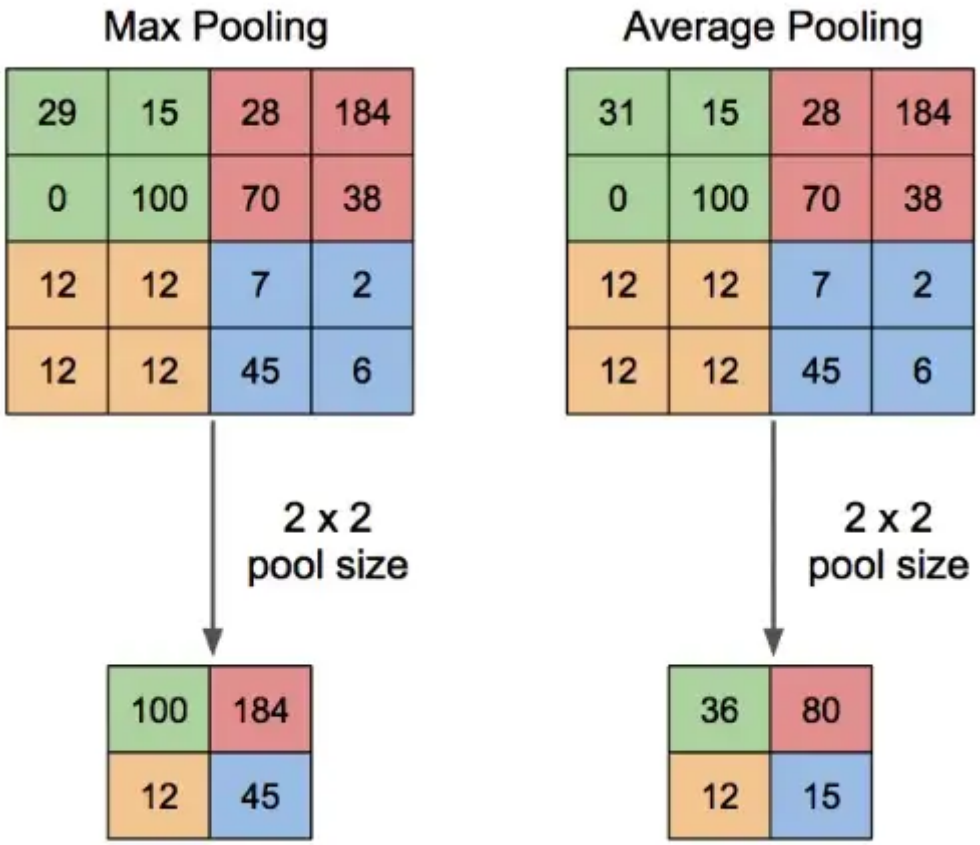
- https://www.datastuff.tech/machine-learning/why-do-neural-networks-need-an-activation-function/

3.2. 첫번째 **Pooling Layer**



그 다음은 Pooling Layer입니다. 우선 **Pooling**이라는 개념이 무엇인지 짚고 넘어가야 합니다.

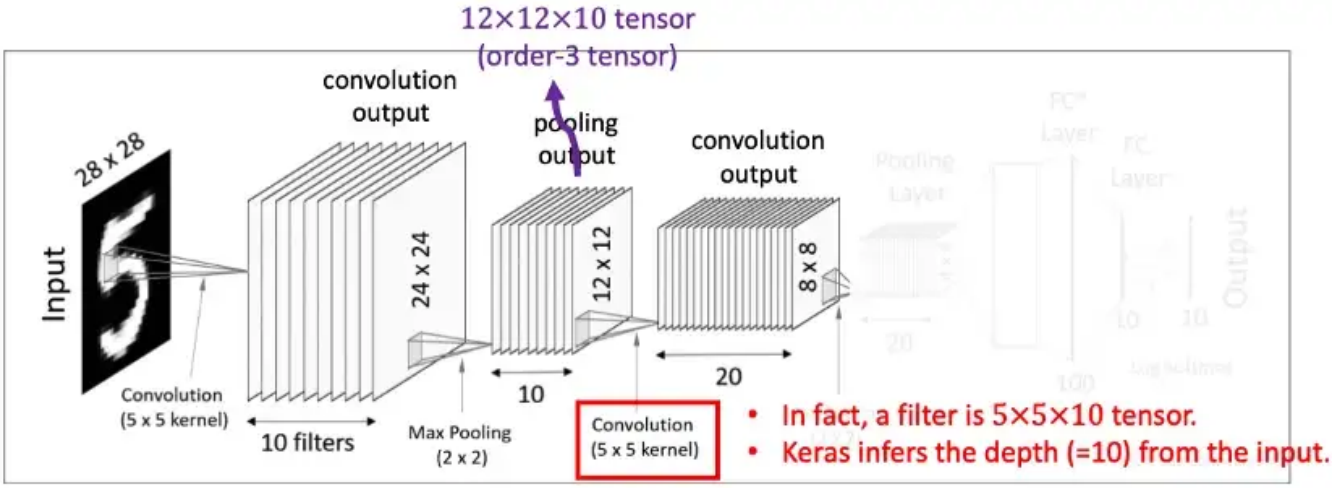
이 전 단계에서 convolution 과정을 통해 많은 수의 결과값(이미지)들을 생성했습니다. 하지만 위와 같이 한 개의 이미지에서 10개의 이미지 결과값이 도출되어버리면 값이 너무 많아졌다는 것이 문제가 됩니다. 때문에 고안된 방법이 Pooling이라는 과정입니다. *Pooling*은 각 결과값*(feature map)*의 *dimentionality*를 축소해 주는 것을 목적으로 둡니다. 즉 correlation이 낮은 부분을 삭제(?)하여 각 결과값을 크기(dimension)을 줄이는 과정입니다.



Pooling에는 대표적으로 두가지 방법으로 Max pooling과 Average pooling이 있습니다. 위 이미지와 같이 Pool의 크기가 2x2인 경우 2x2크기의 matrix에서 가장 큰 값(max)이나 평균값(average)를 가져와 결과값의 크기를 반으로 줄여주게 됩니다.

자, 이제 다시 본론으로 돌아가 그 위의 전체 네트워크 이미지로 돌아가면 Pooling Layer의 결과값이 열개의 12x12 matrics가 된 것을 확인할 수 있습니다.

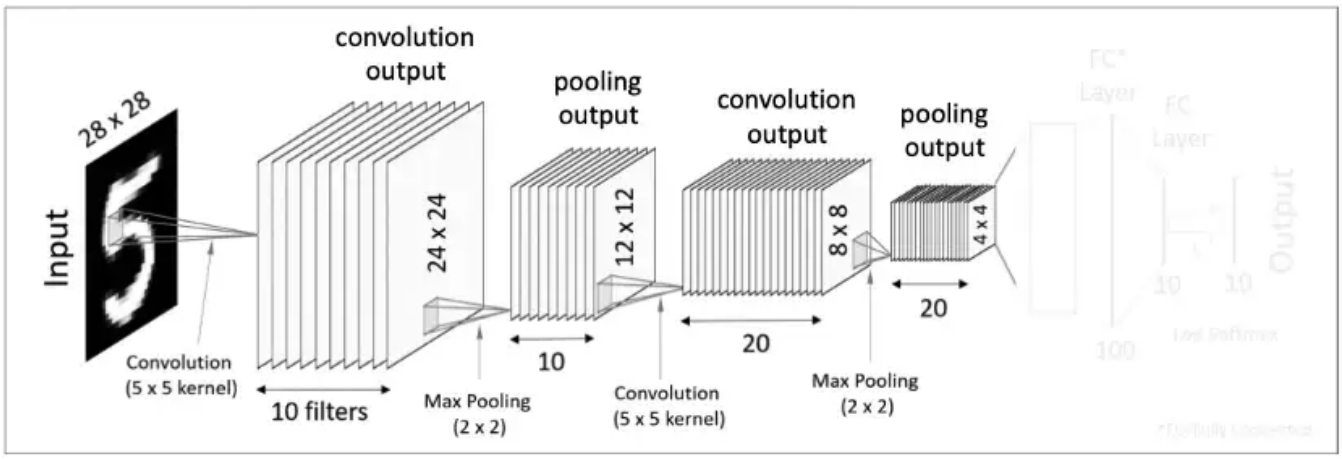
3.3. 두번째 Convolutional Layer



이번 Convolutional layer에서는 텐서 convolution을 적용합니다. 이전의 pooling layer에서 얻어낸 12x12x10 텐서(order-3 tensor)를 대상으로 5x5x10크기의 텐서필터

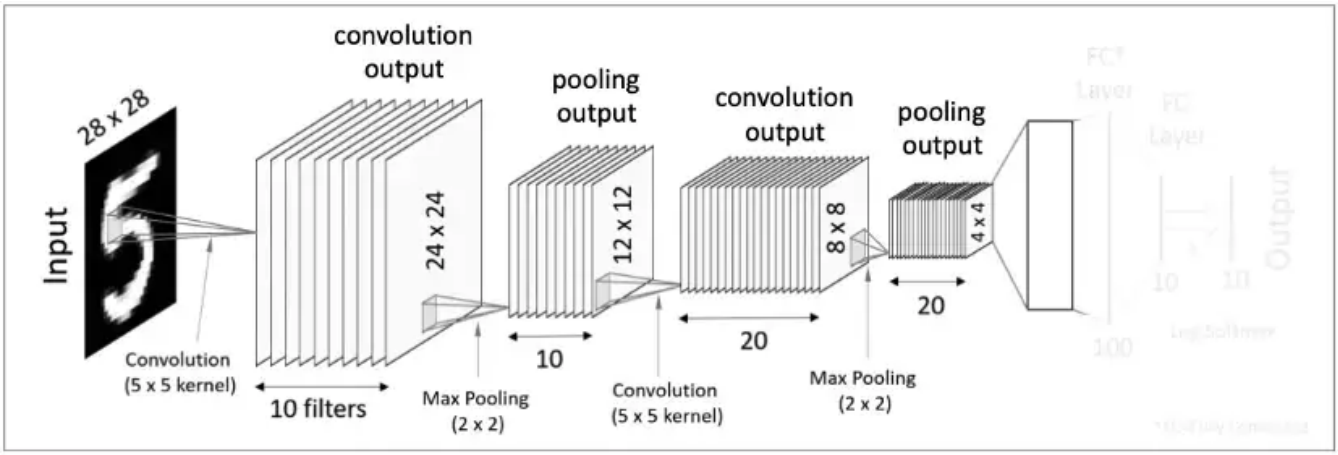
20개를 사용해 줍니다. 그렇게 되면 각각 8x8크기를 가진 결과값 20개를 얻어낼 수 있습니다.

3.4. 두번째 Pooling Layer



두번째 Pooling layer입니다. 전과 똑같은 방식으로 Pooling과정을 처리해주면 더 크기가 작아진 20개의 4x4 결과값을 얻습니다.

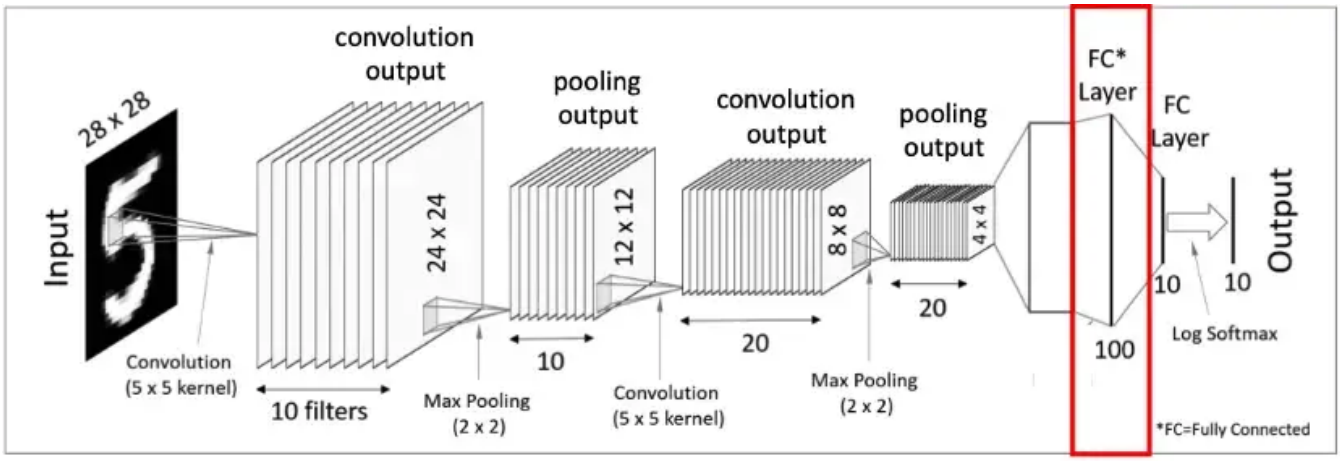
3.5. Flatten(Vectorization)



그 후 이 4x4x20의 텐서를 일자 형태의 데이터로 쭉 펼쳐준다고 생각해 봅시다. 이 과정을 *Flatten* 또는 *Vectorization* 이라고 합니다. 쉽게 말해 각 세로줄을 일렬로 쭉 세워두는 것입니다. 그러면 이것은 320-dimension을 가진 벡터(vector)형태가 됩니다.

그렇다면 왜 이렇게 1차원 데이터로 변형해도 상관이 없을까요? 이 전에 두번째 pooling layer에서 얻어낸 4x4크기의 이미지들은 이미지 자체라기 보다는 입력된 이미지에서 얻어온 특이점 데이터가 됩니다. 즉 1차원의 벡터 데이터로 변형시켜주어도 무관한 상태가 된다는 의미입니다.

3.6. Fully-Connected Layers(Dense Layers)



이제 마지막으로 하나 혹은 하나 이상의 Fully-Connected Layer를 적용시키고 마지막에 Softmax activation function을 적용해주면 드디어 최종 결과물을 출력하게 됩니다.

4. 매개변수(Parameter)와 Hyper-매개변수(Hyper-parameter)

우선 Parameter와 Hyper-parameter, 용어 정리부터 해보도록 하겠습니다(좀더 자세히 알아보기).

모델 매개변수(parameter)는 모델 내부에 있으며 데이터로부터 값이 추정될 수 있는 설정변수(configuration variable)입니다. 모델 하이퍼파라미터(hyper-parameter)는 모델 외부에 있으며 데이터로부터 값이 추정될 수 없는 설정변수 입니다.

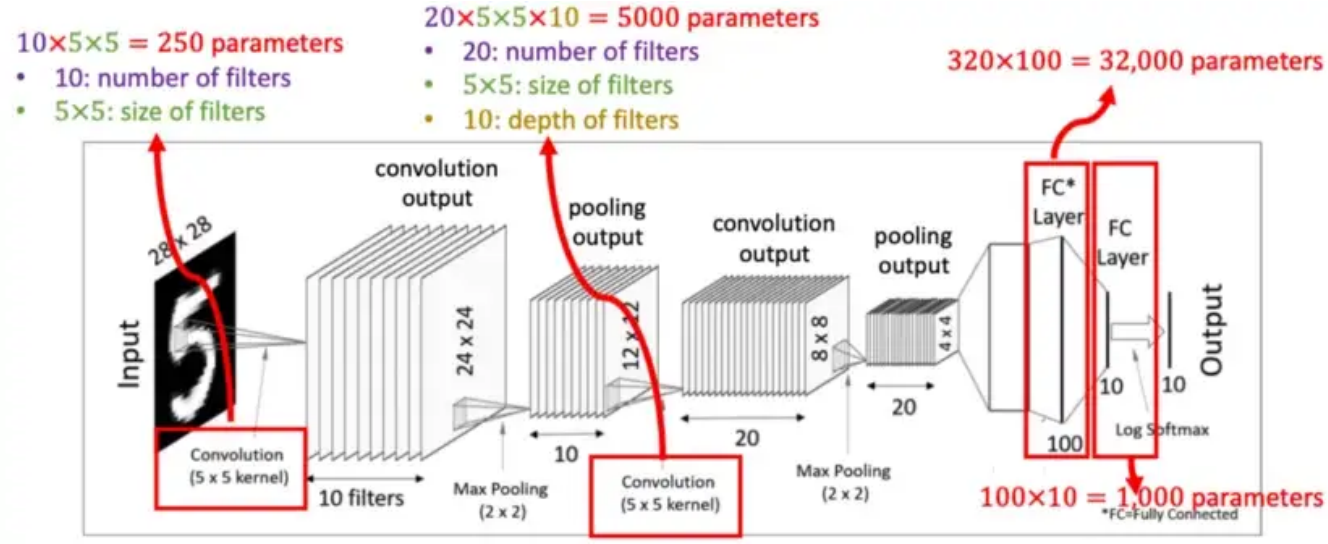
딥러닝의 기본은 파라미터들을 최적의 값으로 빠르고 정확하게 수렴하는 것을 목적으로 합니다. 때문에 어떻게 모델의 파라미터들을 최적화 시키냐 하는 것은 모델 트레이닝의 중요한 포인트입니다.

모델을 학습시킬때 좋은 하이퍼파라미터를 찾는 것 또한 중요합니다. 이것을 '하이퍼파라미터를 튜닝한다(tune haperparameters)'고 표현합니다.

> 이 두 목적을 위한 방법론에 관한 이야기는 무궁무진하므로 또 다른 포스팅에서 다루도록 하겠습니다.

4.1. 학습 가능한 매개변수의 수

이제 우리가 아까 살펴본 CNN 모델의 학습 가능한 매개변수(trainable parameters)는 몇개나 되는지 살펴봅시다.



- 첫 convolution과정에서 5x5크기의 필터 10개를 사용하였으므로 **250**개의 매개변수가 존재합니다.
- Pooling layer에서는 단순히 크기를 줄이는 개념이므로 매개변수가 없습니다.
- 두번째 convolution 과정에서 5x5x10크기의 텐서 스무개를 사용하였으므로 **5,000**개의 매개변수가 추가되었습니다
- 두개의 Fully-connected layer들에서 각각 매개변수 **32,000**개, **1000**개가 추가되었습니다.

결과적으로, 생략한 intercepts 값까지 고려해보면 총 **38,250** 이상의 학습 가능한 매개변수를 갖게 됩니다.

4.2. 초매개변수(Hyper-parameters)

마지막으로 CNN모델에서 튜닝 가능한 하이퍼파라미터는 어떤 것들이 있는지 간단히 살펴보겠습니다.

- Convolutional layers: 필터의 갯수, 필터의 크기, stride값, zero-padding의 유무
- Pooling layers: Pooling방식 선택(MaxPool or AvgPool), Pool의 크기, Pool stride 값(overlapping)
- Fully-connected layers: 넓이(width)
- 활성함수(Activation function): ReLU(가장 주로 사용되는 함수), SoftMax(multi class classification), Sigmoid(binary classification)
- Loss function: Cross-entropy for classification, L1 or L2 for regression

- 최적화(Optimization) 알고리즘과 이것에 대한 hyperparameter(보통 learning rate): SGD(Stochastic gradient descent), SGD with momentum, AdaGrad, RMSprop
- Random initialization: Gaussian or uniform, Scaling

여기서 간단히 알고 넘어가야 할것은 CNN을 학습시킬때 다뤄야 할 hyper-parameter가 이렇게나 많다는 것입니다.

조금 더 큰 그림을 위한 글 링크

이상으로 CNN 기본 개념 설명글을 마치겠습니다. 다음 포스팅에서는 실제로 Keras를 사용해 CNN 모델을 코드로 구현해보고 실제 모델 구현시 알아야할 Trick들에는 무엇이 있는지 살펴보도록 (노력)하겠습니다.

Written by: Mijeong Ban

halfundecided - Overview

Arctic Code Vault Contributor Pro Dismiss Sign up for your own profile on GitHub, the best place to host code, manage...

github.com

Deep Learning Machine Learning 머신러닝 딥러닝