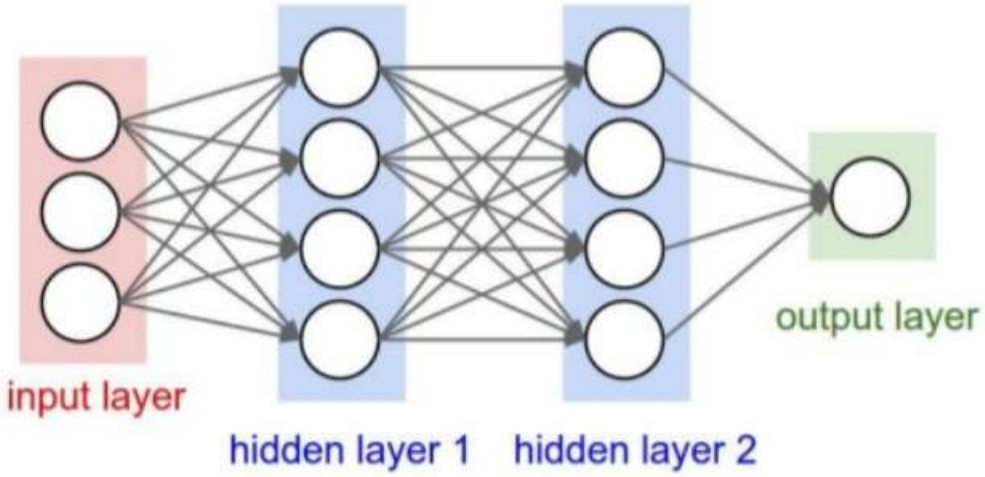
0119 CNN

1. Intro to CNN

- CNN은 공간적 / 지역적 정보를 유지한 상태로 학습 가능하도록 만든 모델이다.
    - 1차원 배열 형태만 input으로 받던 DNN의 구조적 한계로 인한 공간적/지역적 정보 손실을 방지
- CNN의 구성: Overview
    - 이미지 특징 추출 → Convolution Layer와 Pooling Layer를 여러 겹으로 쌓아 구성하며, 학습을 통해 데이터의 특성을 뽑아낸다.
    - 클래스 분류 → 계산된 특징 데이터를 1차원 텐서로 변환한 뒤 Fully Connectecd Layer에 정보를 전달, 활성 함수를 적용해 예측값을 계산한다.
- 기본 용어 정리
    - 채널 → 이미지 픽셀 값의 표현
        - 이미지 텐서는 높이 x 너비 x 채널의 3차원으로 표현된다.
        - 흑백 사진은 하나의 실수만 필요 (=1개의 채널로 구성)
        - 컬러 사진은 각 픽셀에 3개의 실수 필요 (=3개의 채널로 구성)
    - 필터 또는 커널 (filter / kernel) → 특징을 찾아내기 위한 공용 파라미터
    - 활성화 함수 → 입력 신호의 합을 출력 신호로 변환하는 비선형 함수
        - 선형이 아닌 데이터의 패턴과 관계를 학습할 수 있게 돕는다.
        - ReLU, Sigmoid, Tanh, Leaky ReLU 등 여러 종류가 있지만, CNN에서 주로 사용하는 활성화 함수는 ReLU이다.

$$f(x) = max(0, x)$$

- Fully Connected Layer (완전 연결 층)
  - CNN의 마지막 단계에서 주로 사용되는 신경망 층으로, 각 뉴런이 이전 층의 모든 뉴런과 연결되어 있다는 특징이 있다. (즉, 이전 층의 특징을 모두 사용한다)



2. 이미지 특징 추출하기

1) Convolution Layer + Activation Function

- 입력 데이터에 필터를 적용해 특징 맵을 생성하고, 만들어진 특징 맵에 활성화함수를 적용한다.

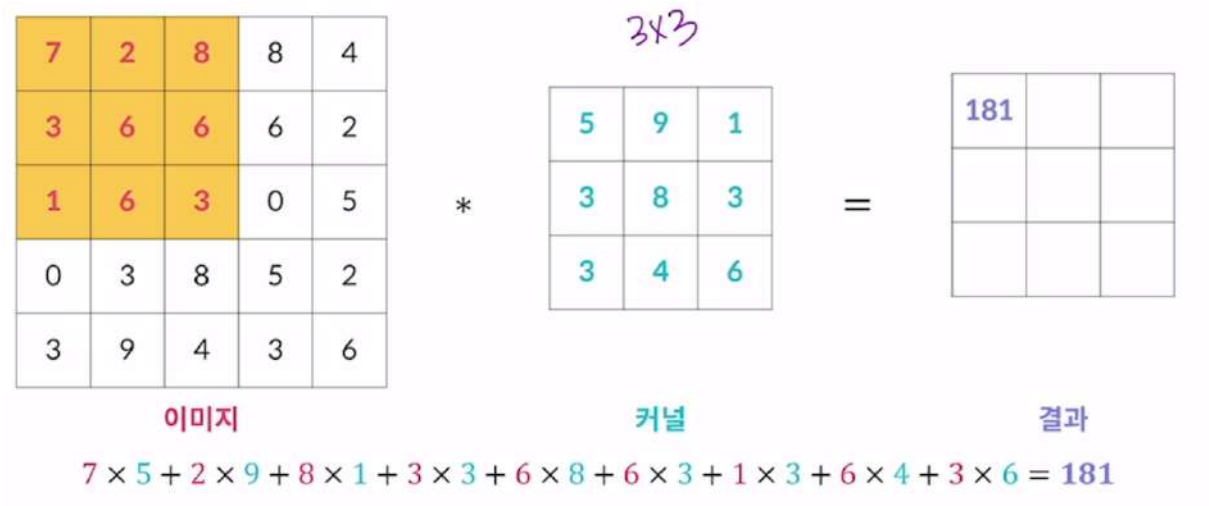
> 📌 **특징 맵**은 어떻게 생성할까?

(1) 하나의 Conv layer에는 이미지의 채널 수만큼 필터가 존재한다.

- 각 채널은 각각에 할당된 필터를 사용한다.

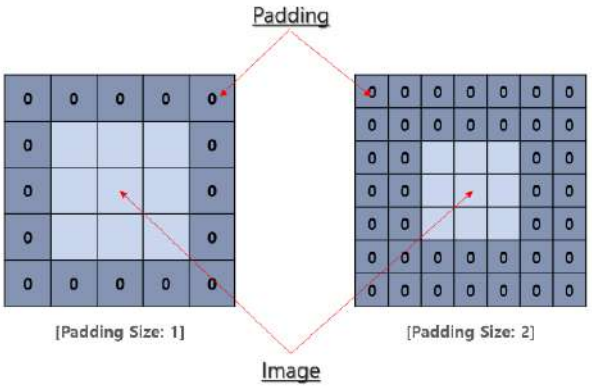
(2) 이미지의 왼쪽 상단부터 시작해 **필터를** 오른쪽으로 한 칸씩 (혹은 여러 칸씩) **이동하며 연산**한다.

(3) 필터를 적용하는 각 위치에서는 **필터와 이미지 해당 부분의 element-wise multiplication** (요소별 곱)을 수행하고, **결과를 모두 합산**해 하나의 숫자를 생성한다.



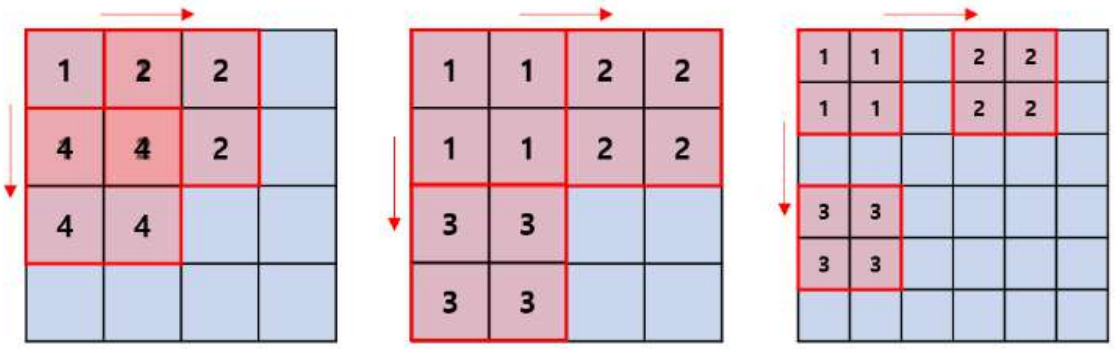
🤨 결국 이미지와 필터의 연산을 통해 나온 출력의 크기는 작아진다. 그렇다면, 출력 크기가 작아질 때마다 가장자리 정보들이 손실되는 것은 아닐까?

- 이를 해결하기 위해 등장한 것이 **Zero-Padding**
  - 0으로 구성된 테두리를 가장자리에 감싸 이미지 사이즈를 임의로 늘려주자. (padding 할 값은 hyper-parameter로 어떤 값을 사용할지 직접 결정하는 것도 가능하다.)



🤨 필터를 꼭 한 칸씩 이동해야 할까? 연산이 너무 오래 걸리는 것 같아 이동 간격을 조절하고 싶다!

- **Stride**를 통해 필터 적용 간격을 조정할 수 있다.



🤨 Padding과 Stride를 적용했을 때 최종 output의 사이즈를 미리 계산해볼 수 있을까?

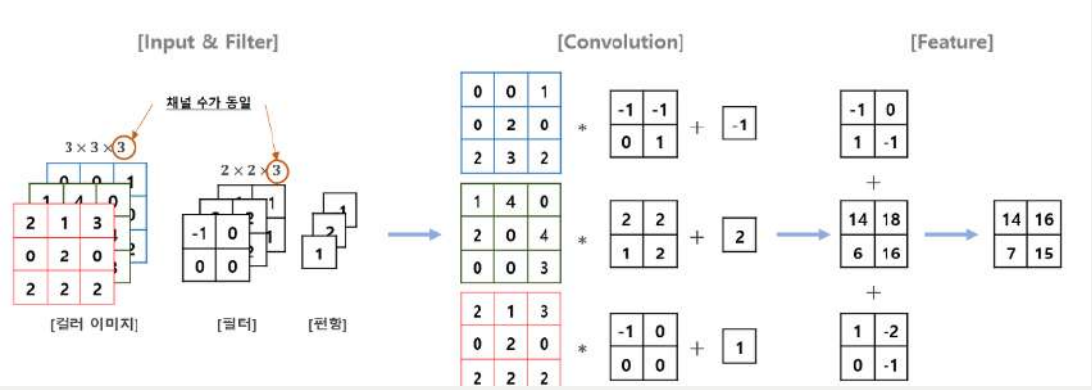
$$(OH, OW) = \left(\frac{H + 2P - FH}{S} + 1, \frac{W + 2P - FW}{S} + 1 \right)$$

- (H; 높이, W; 너비) : 입력 크기
- (FH , FW) : 필터 크기
- S : 스트라이드
- P : 패딩

(4) 생성한 하나의 결과를 특**징 맵의 해당 위치에 입력**한다.

📌 컬러 이미지의 Feature Map

컬러 이미지의 경우 채널이 3개이므로 필터 역시 채널이 3개인 필터를 사용한다. 각 채널별 convolution 연산을 수행해 feature map을 뽑아내며, 각각의 결과를 더해 하나의 feature map을 생성한다.

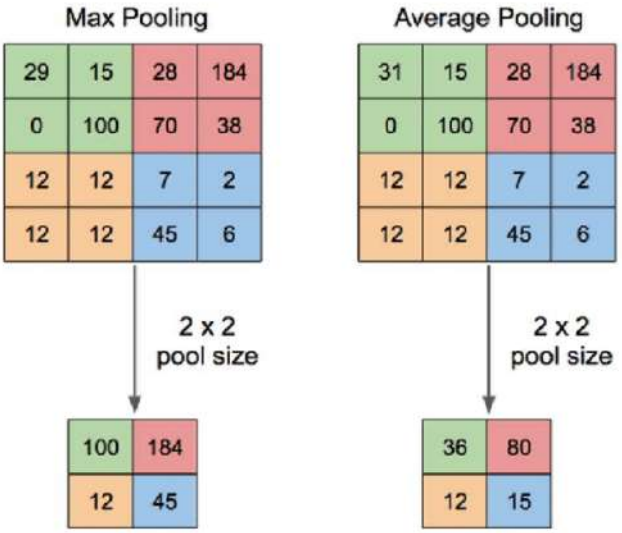


📌 Activation Function 적용하기

- Convolution Layer의 결과인 Feature Map에 Activation function (ex. ReLU)을 적용해 픽셀에 특징이 있는지 없는지와 관련된 비선형 값으로 바꿔준다.

2) Pooling Layer

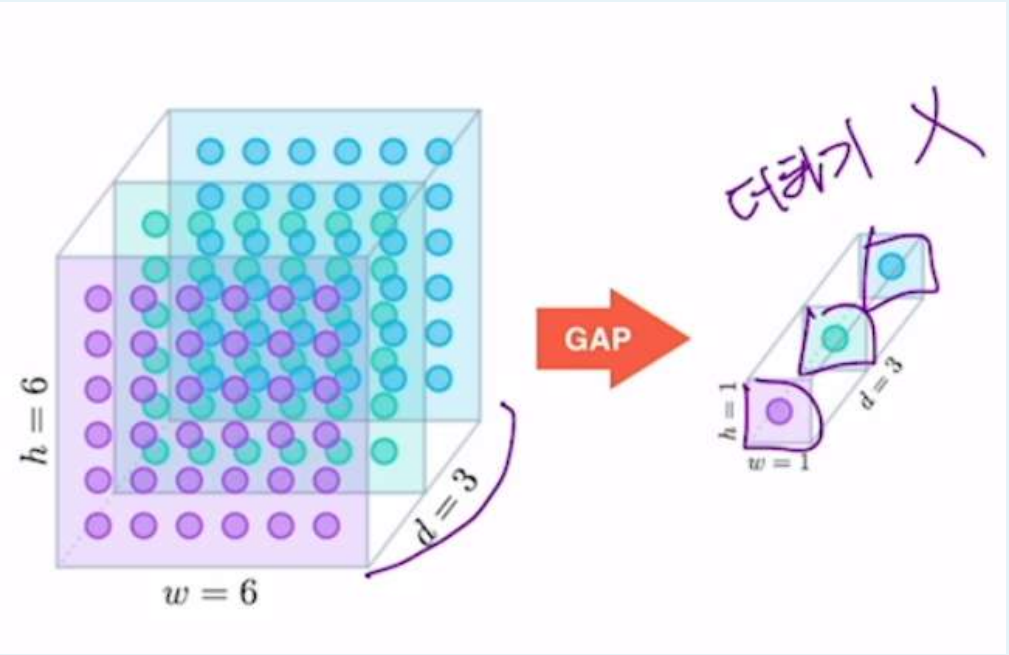
- 활성화 함수를 거쳐 나온 Feature Map의 사이즈를 줄여 특징을 뽑아낸다.
- Feature map을 m x n 사이즈로 조각내고, 각 영역마다 조건에 맞는 값을 뽑아낸다.



- Max Pooling → m x n 사이즈의 각 영역에서 가장 큰 값을 뽑는다.
- Average Pooling → m x n 사이즈의 영역에서 평균값을 뽑는다.

🤨 Feature map 의 채널이 여러 개인 경우의 Pooling 연산

- 채널 별로 연산을 수행하되, Convolution처럼 더하지 않는다.



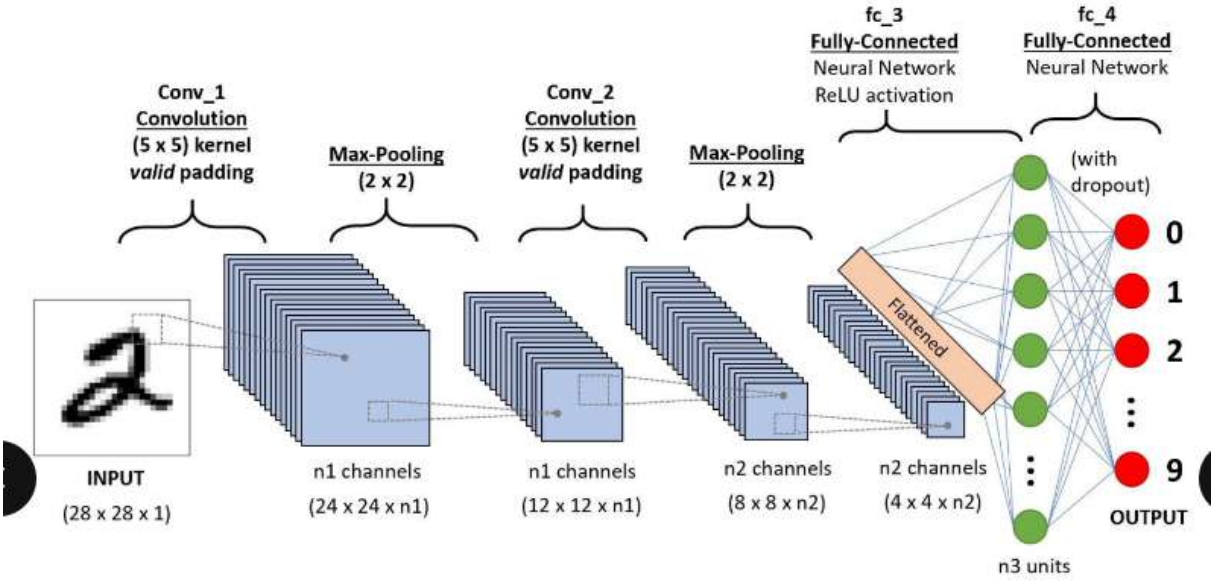
3. 클래스 분류하기

- CNN 네트워크의 마지막에서는 클래스 분류를 위해 **Fully Connected Layer에 전달**하는 과정을 거친다. 이 때, Fully Conntected Layer에 전달하기 위해서는 **데이터를 1차원 텐서로 만드는 Flatten 과정**을 거쳐야 한다.

🤨 특징을 담은 데이터를 1차원으로 만들면 DNN처럼 정보가 소실될 수도 있지 않나요?

→ 이미 지금까지의 연산을 통해 각 데이터는 특징 데이터가 되므로 1차원 텐서로 변환해줘도 문제 없다 !

- 하나 이상의 FC Layer에 텐서를 적용시키고, Log Softmax 등의 Activation 함수를 적용해 최종 예측값을 계산하면 클래스 분류 완료 !



4. CNN 구조를 기반으로 학습 과정 이해하기

1. 파라미터 초기화 : 가중치와 편향을 무작위로 초기화 (혹은 특정 전략 따르기)
2. Forward Propagation
    - Convolution layer + Activation Function을 통해 이미지에 필터를 적용한 결과인 특징 맵을 생성한다.
    - 필요하다면 Pooling을 통해 특징 맵의 크기를 줄인다.
    - Flatten한 특징 정보를 Fully Connected Layer에 input으로 넣어 최종 출력을 생성한다.
3. Loss 계산
    - 신경망의 분류 결과와 실제 정답 사이의 오차 계산을 위해 손실 함수를 사용한다. (분류 문제에서는 보통 Cross-Entropy Loss 사용)
4. Backward Propagation
    - 손실 함수의 Gradient를 계산하고, 이를 이용해 가중치와 편향에 대한 업데이트를 결정한다.
    - 역전파의 기본 규칙은 Chain Rule이다.
5. 가중치 업데이트
    - 계산된 Gradient를 사용해 신경망의 가중치와 편향을 업데이트한다.
    - SGD와 같은 optimizer를 사용한다.