1. 합성곱 신경망

날짜	@2023년 12월 24일

▼ 목차

컴퓨터비전

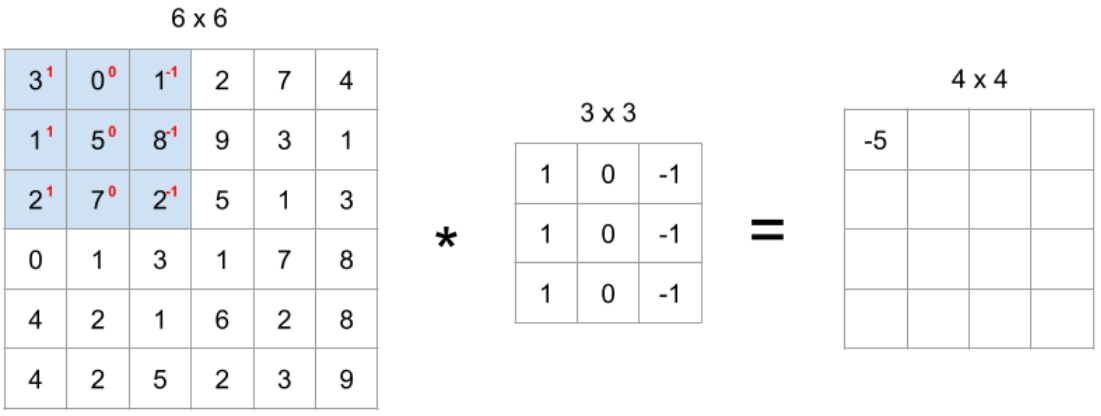
- 얼굴 인식, 예술 등 다양한 분야에 응용되며 이미지 분류, 객체 인식, 신경망 스타일 변형 등의 문제를 다룸

- 컴퓨터비전 알고리즘의 발전은 새로운 비전 관련 어플리케이션을 창출해낼 뿐만 아니라, 자연어 처리 등 다른 분야에도 영향을 줌
- 컴퓨터비전에서는 입력 데이터가 아주 크다는 것이 장애물이지만, 합성곱 연산을 통해 이를 어느 정도 해결할 수 있음

모서리 감지 예시

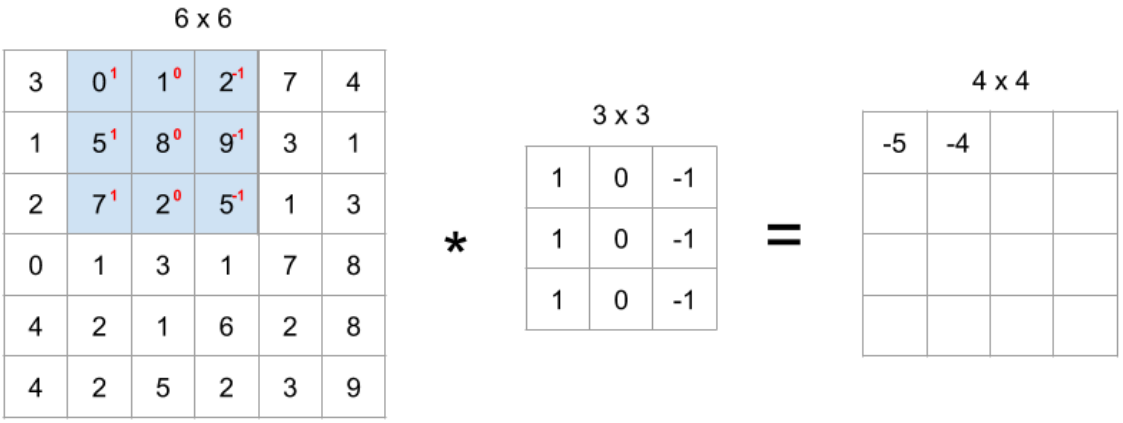
1️⃣ 합성곱 연산

- 이미지는 `높이×넓이`로 표현할 수 있음
- 합성곱 연산의 진행 과정



3x1 + 0x0 + 1x-1 + 1x1 + 5x0 + 8x-1 + 2x1 + 7x0 + 2x-1 = -5

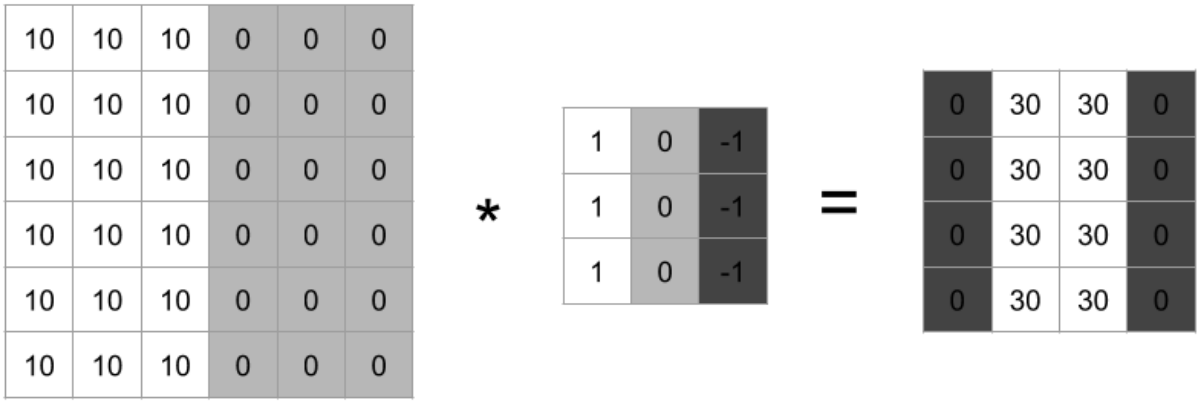
  - 원래 이미지(위의 6×6 행렬)와 필터(또는 커널, 위의 3×3 행렬)를 이용함
  - 각각의 원소를 곱한 후 전부 더함



0x1 + 1x0 + 2x-1 + 5x1 + 8x0 + 9x-1 + 7x1 + 2x0 + 5x-1 = -4

- 필터(커널)를 한 칸씩 이동시키며 합성곱 연산을 진행하여 새로운 행렬을 만듦
  - 위 예시의 경우 최종적으로 4×4 형태의 새로운 행렬이 만들어짐

2️⃣ 수직 윤곽선 탐지



- 위 그림의 왼쪽 이미지에서 10과 0 사이의 경계선이 수직 윤곽선
- 필터를 통과시켜 합성곱 연산을 하게 되면 밝은 부분이 중앙으로 나타나게 되며, 이것이 원래 이미지의 수직 경계선에 해당하는 부분임
  - 위와 같이 원래 이미지의 크기가 작은 경우 검출된 경계선이 조금 더 두껍게 나타날 수 있음

더 많은 모서리 감지 예시

1️⃣ 필터의 종류

- 윤곽선 탐지를 위한 다양한 필터가 있음 (sobel, scharr 등)
- 최근 딥러닝에서는 임의의 숫자로 필터를 만든 다음 역전파를 통해 모델이 스스로 학습하여 문제에 적합한 필터를 만드는 방법을 사용함

1	0	-1
1	0	-1
1	0	-1

vertical

1	0	-1
2	0	-2
1	0	-1

sobel

3	0	-3
10	0	-10
3	0	-3

scharr

1	1	1
0	0	0
-1	-1	-1

horizontal

1	2	1
0	0	0
-1	-2	-1

sobel

3	10	3
0	0	0
-3	-10	-3

scharr

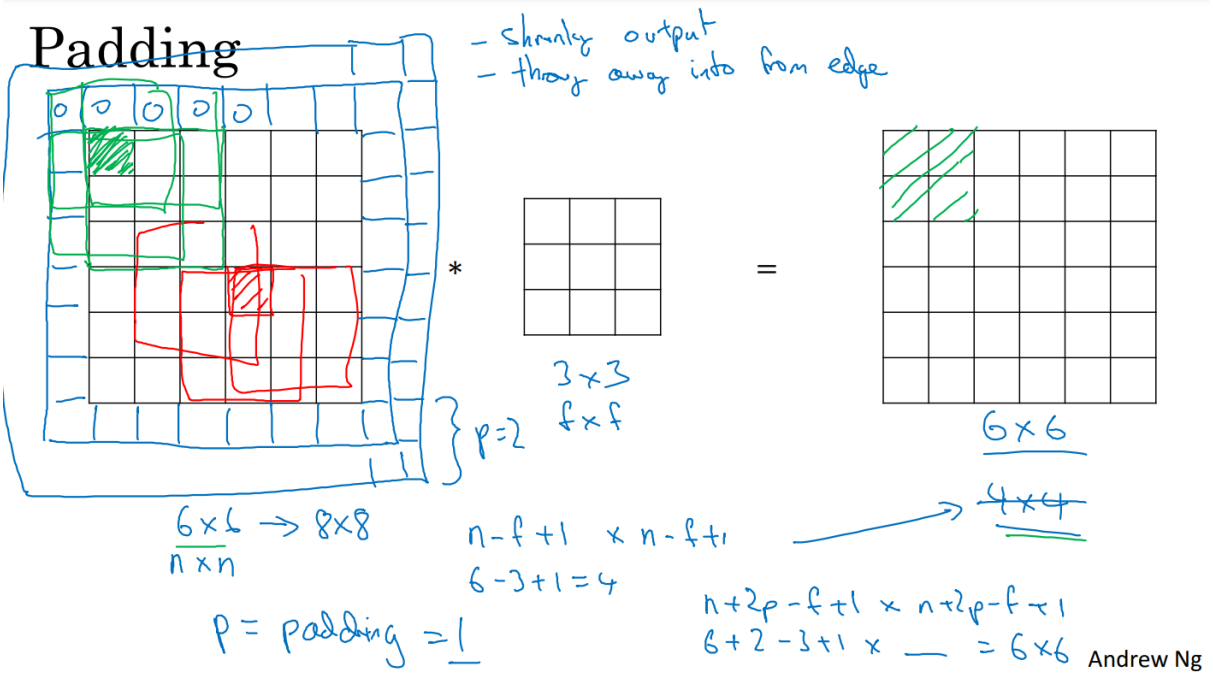
패딩(Padding)

1️⃣ 합성곱 연산의 문제점

- 🔹 합성곱 연산을 여러 차례 진행할수록 이미지는 계속 축소되는데, 이때 가장자리에 위치한 픽셀은 단 한 번만 사용되므로 이미지의 윤곽에 해당하는 정보가 버려짐

2️⃣ Padding

- 이미지 주위에 경계를 추가하여 덧대는 방법으로, 이미지 크기를 증가시킴
  - 일반적으로 경계에는 숫자 0을 사용함



- 최종 이미지 크기: $(n+2p-f+1) \times (n+2p-f+1)$
  - n: 이미지 크기
  - p: 패딩 크기
  - f: 필터 크기
- 일반적으로 필터의 크기는 홀수로 설정함
  - f가 짝수인 경우 패딩이 비대칭이 됨
    - 홀수일 경우에는 합성곱 연산 시 동일한 크기로 패딩을 더할 수 있는 반면, 짝수일 경우 왼쪽과 오른쪽을 다르게 패딩해야 하므로 번거로움
  - f가 홀수인 경우 중심 위치가 존재하여 활용이 편리함

스트라이드(Stride)

1️⃣ Stride

- 필터의 이동 횟수
  - 값을 설정할 경우 필터는 스트라이드의 수만큼 이동하며 연산을 진행함
- 최종적으로 출력되는 행렬의 크기: $(\frac{n+2p-f}{s}+1) \times (\frac{n+2p-f}{s}+1)$
  - 소수점 아래 숫자가 존재하는 경우 내림

- 일반적으로는 정수가 되도록 패딩과 스트라이드 수치를 필터에 맞춤

2️⃣ Cross-correlation vs. Convolution

> 🔹 지금까지 배운 합성곱은 수학적으로 엄밀히 말하면 교차상관(Cross-correlation)이지만, 딥러닝에서는 관습적으로 합성곱(Convolution)이라고 부름

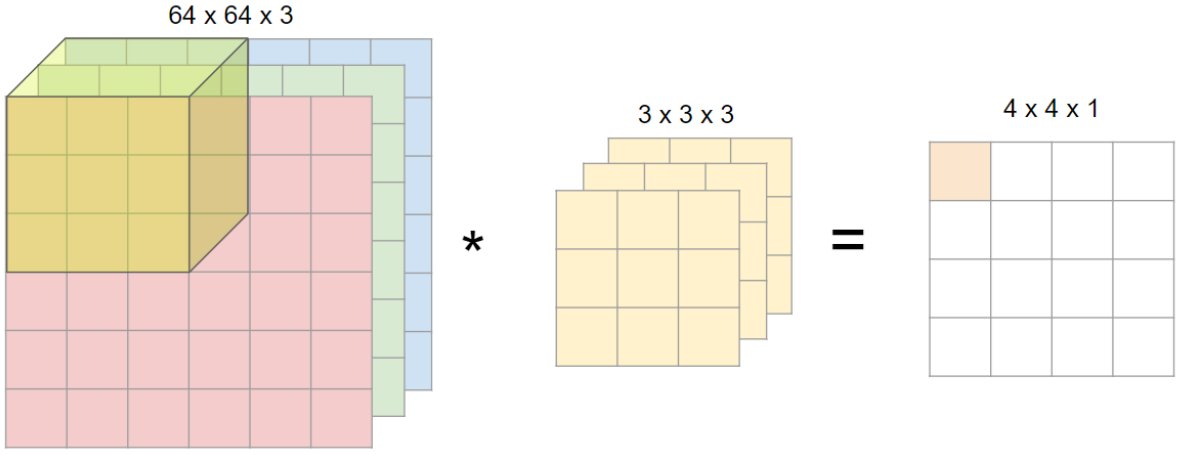
- 일반적으로 수학에서 정의하는 합성곱(Convolution)은 합성곱을 하기 전 필터를 가로축과 세로축으로 뒤집는 연산을 진행함
- 그러나 딥러닝에서는 해당 연산을 생략함
  - 해당 연산 과정은 신호처리에서는 유용하지만 심층 신경망 분야에서는 아무런 영향이 없기 때문에 생략이 가능함

입체형 이미지에서의 합성곱

1️⃣ 입체형 이미지

- 이미지에 색상(RGB)이 들어가면 입체형으로 변하게 되며, 차원이 하나 증가함
- 즉 `높이×넓이×채널`로 변하게 되는데, 이때 채널은 색상 또는 입체형 이미지의 깊이를 뜻함
- 이에 따라 합성곱에 사용되는 필터 또한 각 채널별로 하나씩 증가하게 됨

2️⃣ 입체형 이미지의 합성곱 계산



- 모든 채널의 합성곱 연산을 더해주는 방식으로 진행됨

- 각 채널별로 필터를 다르게 설정할 수 있음
- 패딩과 스트라이드가 없다고 가정했을 때, 최종적으로 출력되는 행렬은 $(n \times n \times n_c) * (f \times f \times n_c) = (n-f+1) \times (n-f+1) \times n_{c'}$ 형태가 됨
  - n: 이미지의 크기
  - n_c: 채널의 개수
  - f: 필터의 크기
  - $n_{c'}$: 사용된 필터의 개수

합성곱 네트워크의 한 계층 구성하기

1️⃣ Example of a Layer

- 합성곱 신경망의 한 계층은 아래와 같이 구성됨
  - 합성곱 연산 → 편향 추가 → 활성화 함수
  - 활성화 함수는 비선형성을 적용하기 위해 사용되며, 일반적으로 ReLU를 사용함

2️⃣ Summary of Notation

- 기본적인 표기법
  - l: l번째 계층
  - $f^{[l]}$: 필터의 크기
  - $p^{[l]}$: 패딩의 양
  - $s^{[l]}$: 스트라이드 크기
  - n_H: 이미지의 높이
  - n_W: 이미지의 넓이
  - n_c: 채널의 수
- 계산되는 것
  - l번째 층의 이전 층 $(l-1)$의 이미지의 크기
    - $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$
  - 그 결과로 나오는 이미지의 크기

        - $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$
    - l번째 층의 높이 혹은 넓이의 크기 연산
        - $n_H^{[l]} = \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$
        - $n_W^{[l]} = \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1$

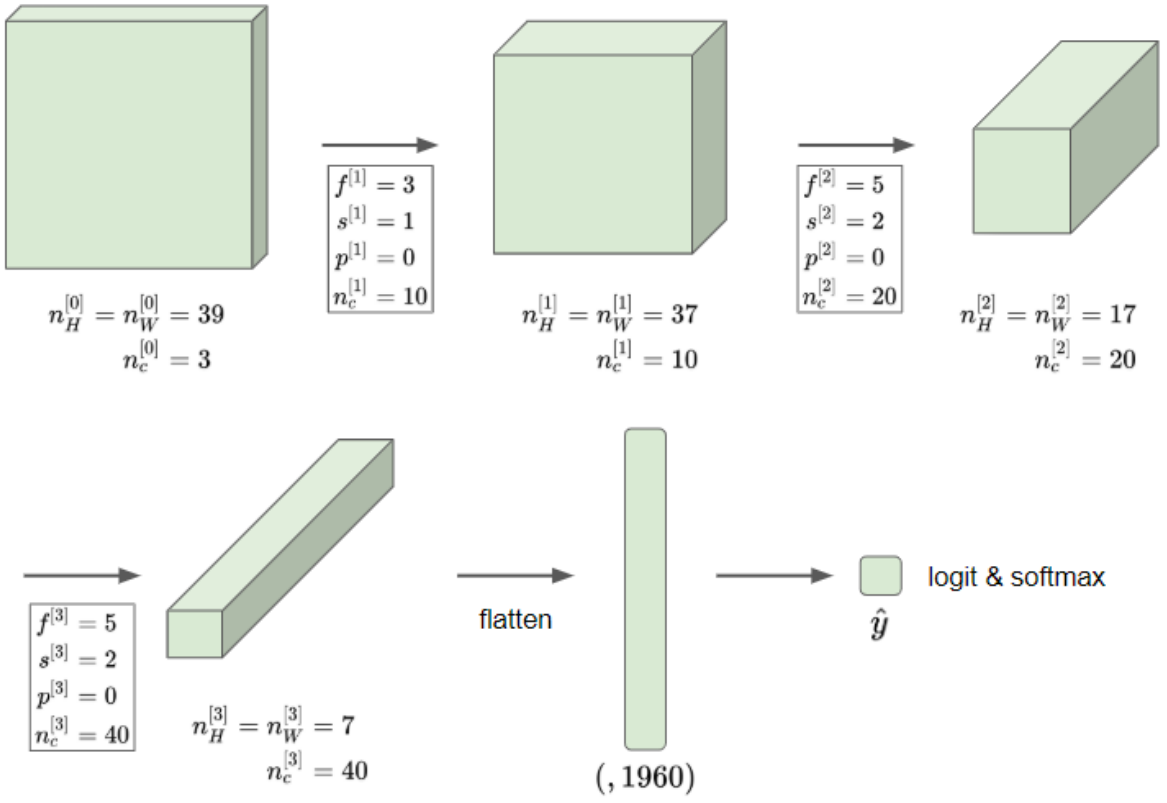
3️⃣ 계산 과정 정리

- 크기가 $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$인 필터 $n_c^{[l]}$개로 합성곱 연산을 진행하게 됨
- 활성화 함수를 거쳐 l번째 층의 결과값이 계산됨
- 합성곱 연산에 사용되는 변수의 개수는 $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
    - 편향을 추가할 경우 $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]} + n_c^{[l]}$가 됨
- 기존의 단순 신경망을 사용한다면 가중치 행렬 $W^{[l]}$의 크기는 다음과 같음
    - $(n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}) \times (n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]})$
        - 예를 들어, 패딩이 없고 스트라이드가 1이라고 가정할 경우 `28×28×3` 이미지를 동일한 `5×5` 필터 20개를 사용하여 계산한다면 `24×24×20` 크기의 결과가 출력됨
        - 이때 합성곱 연산에 필요한 총 변수의 개수는 5×5×3×20+20 = 1520가 됨
        - 단순 신경망을 사용하여 같은 크기의 결과를 나타내려면 (28×28×3)×(24×24×20)+(24×24×20) = 27,106,560만큼의 변수가 필요함

간단한 합성곱 네트워크 예시

1️⃣ 합성곱 신경망의 구조

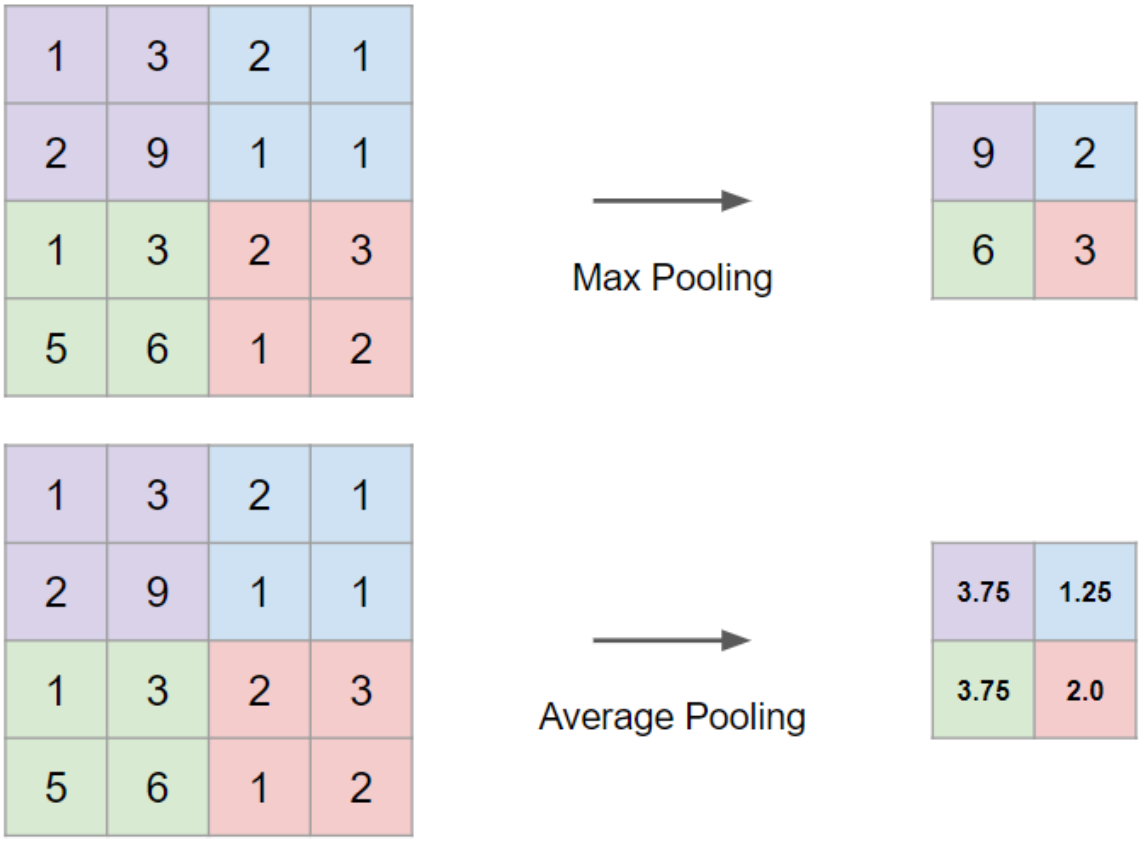
- 합성곱 신경망의 크기는 깊어질수록 점점 줄어듦
- 대부분의 신경망은 합성곱 층(convolution layer), 풀링 층(pooling layer), 완전 연결 층(fully connected layer)으로 구성됨



풀링(Pooling)층

1️⃣ Pooling Layer

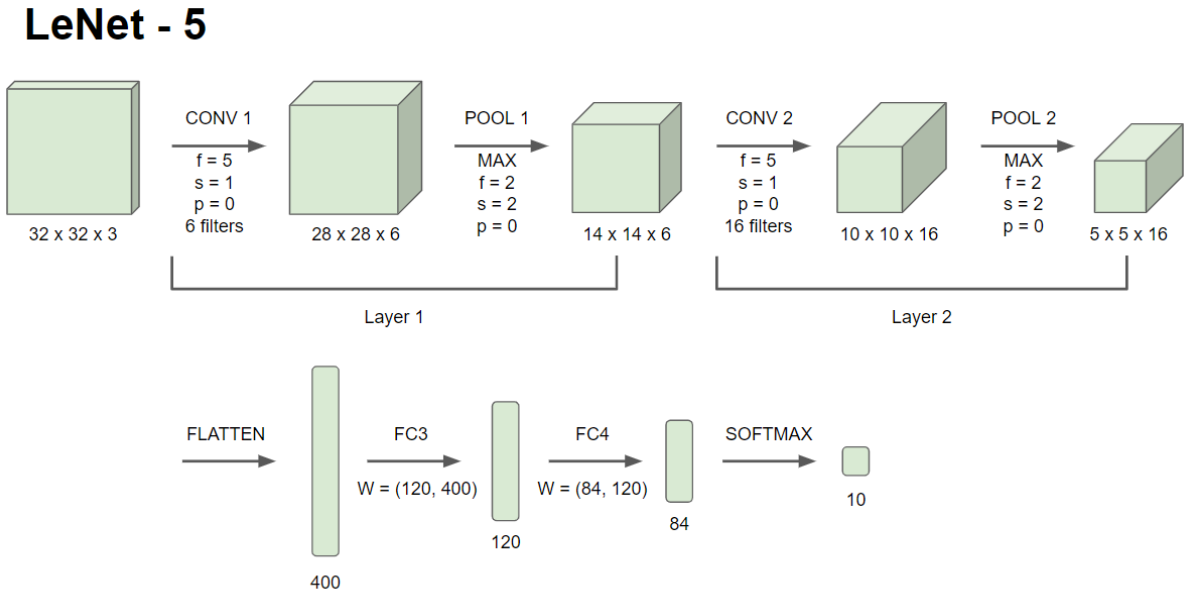
- 합성곱 신경망에서는 풀링 층을 사용해 표현의 크기를 줄임으로써 계산속도를 줄이고 특징을 더 잘 검출해낼 수 있음
- 최대 풀링(Max Pooling)과 평균 풀링(Average Pooling)이 있음
  - 일반적으로 최대 풀링을 사용함
- 최대 연산은 필터의 한 부분에서 이미지의 특징이 검출되면 높은 수를 남기고 그렇지 않으면 다른 최댓값들에 비해 상대적으로 작아지게 하여 특징을 더 잘 남길 수 있도록 함



- 위 예시의 최대 풀링은 필터 크기가 2, 스트라이드가 2, 패딩이 없는 필터를 합성곱 연산이 아닌 최대 연산을 하는 것과 같음
- 이전 강의의 공식을 적용할 수 있으므로 `4×4` 이미지의 출력 결과는 다음에 따라 `2×2`가 됨
  - 결과의 크기: $\frac{4+2\times 0-2}{2}+1=2$

CNN 예시

- 고전적인 합성곱 신경망 LeNet-5의 구조



- 합성곱 신경망의 분야에서는 다음 두 종류의 관습이 존재함
  - 합성곱 층과 풀링 층을 하나의 층으로 간주함
  - 합성곱 층과 풀링 층을 별개의 층으로 간주함
- 위 예시에서는 풀링 층에 학습해야 할 변수가 없으므로 합성곱 층과 풀링 층을 하나로 간주함

왜 합성곱을 사용할까요?

1️⃣ 합성곱 신경망의 장점

- 변수를 적게 사용할 수 있음
  - 예를 들어 `32×32×3` 이미지를 `5×5` 필터 6개를 통해 `28×28×6`의 이미지로 합성곱 연산을 했을 경우, 필요한 변수의 개수는 5×5×3×6+6 = 456
  - 일반적인 신경망에서 필요한 변수의 개수는 3,072×4,704+4,704 ≒ 약 1400만
- 이동 불변성을 포착하는 데 용이함
  - 이미지가 약간의 변형이 있어도 이를 포착할 수 있음

2️⃣ 필요한 변수의 개수가 적은 이유

- 변수 공유

    - 원래 이미지의 한 부분에서 이미지의 특성을 검출하는 필터가 다른 부분에도 똑같이 적용될 수 있음
- 희소 연결
    - 출력값이 이미지의 일부(작은 입력값)에만 영향을 받으며, 이에 따라 과대적합을 방지할 수 있음

출석퀴즈 오답노트

▼ 8. 합성곱 신경망이 변수를 적게 사용하는 이유는?

- 희소 연결 (O)
- 변수 공유 (O)
- 풀링 층 활용 (X)

▼ 9. 2개의 채널로 이루어진 이미지를 5x5 필터 6개를 통해 28x28x6의 이미지로 합성곱 연산하는 경우, 필요한 변수의 개수는?

- 필요한 변수의 개수는 5 * 5 * 2 * 6 + 6으로 총 **306개**이다.