2. 케이스 스터디

1. 왜 케이스 스터디를 하나요?

- 합성곱 신경망을 구축
  - 효율적인 신경망 구조 파악 필요.

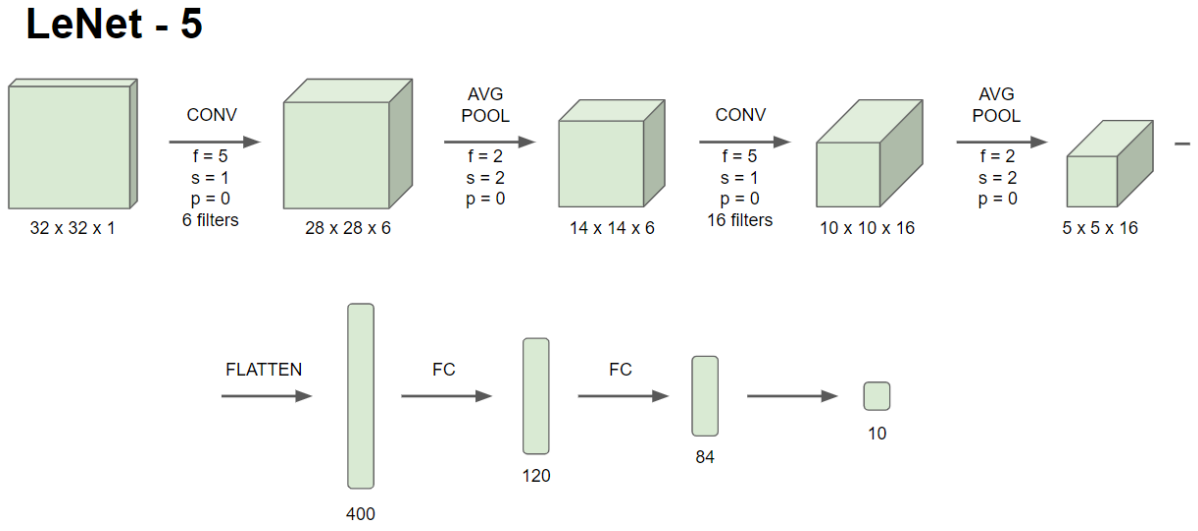
  → 하나의 컴퓨터 비전 작업에서 잘 작동한 구조가 다른 작업에도 유용하고 잘 작동하기 때문

- **대표적인 신경망**
  - LeNet - 5
  - AlexNet
  - VGG
  - ResNet
  - Inception

2. 고전적 네트워크들

1. **LeNet - 5**

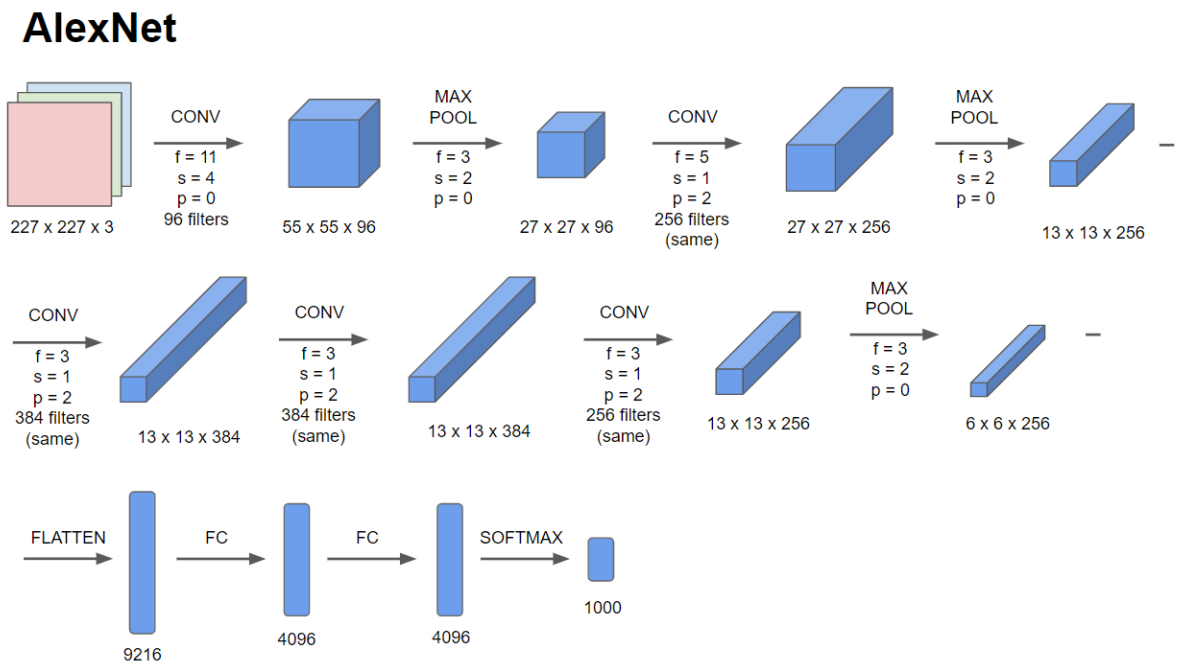
- 목적: 흑백으로 된 손글씨 인식
- 구조



- 상대적으로 적은 변수를 가짐
- 풀링층 뒤에 비선형함수를 적용
- 비선형함수도 ReLU 가 아닌 Sigmoid 를 적용

2. **AlexNet**

- 목적: 이미지를 1000개에 해당하는 클래스로 분류하는 것
- 구조



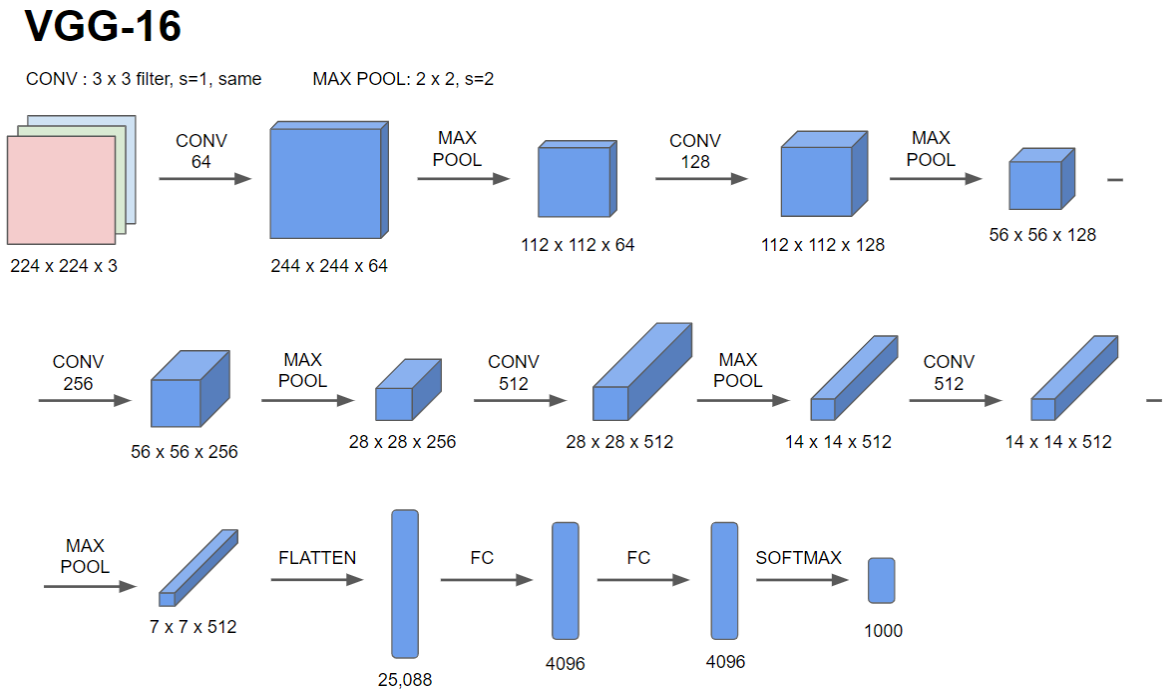
- LeNet 에 비해서 굉장히 많은 변수를 가짐
- “합성곱을 같게 가져간다 (same)” = 이전 층의 높이와 넓이를 같게 만드는 패딩을 가진다

3. **VGG-16**

- AlexNet보다 간단한 구조
- 특징
    - 모든 합성곱 연산은 **3 x 3 의 필터를 가지고 패딩 크기는 2, 스트라이드는 1로 하고, 2 x 2 픽셀씩 최대 풀링**하는 것.

- 산출값의 높이와 넓이는 매 최대 풀링 마다 1/2씩 줄어들며, 채널의 수는 두배 혹은 세배로 늘어나게 만든다.
- 훈련시킬 변수의 개수가 많아 네트워크의 크기가 커진다는 단점

- 구조

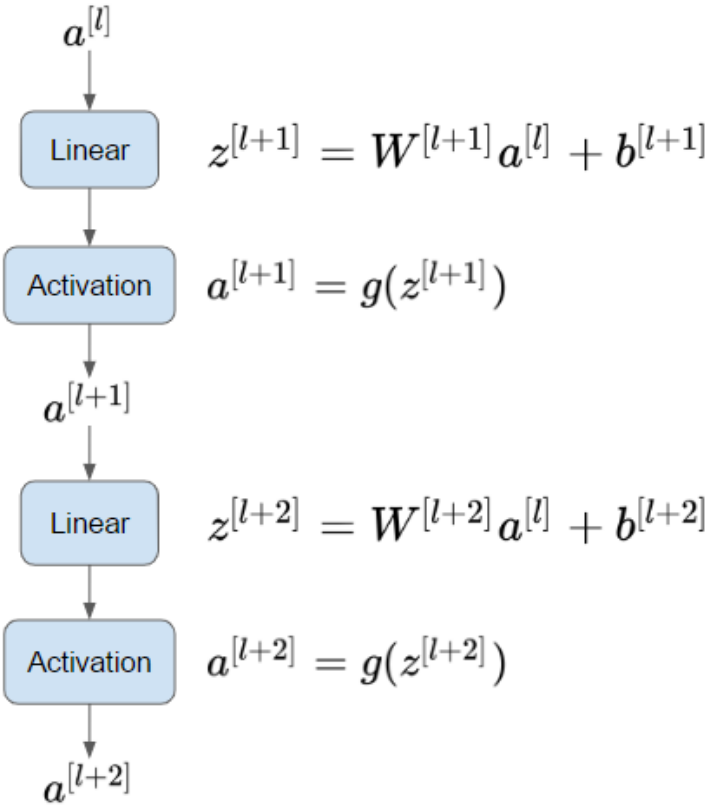


3. Resnets

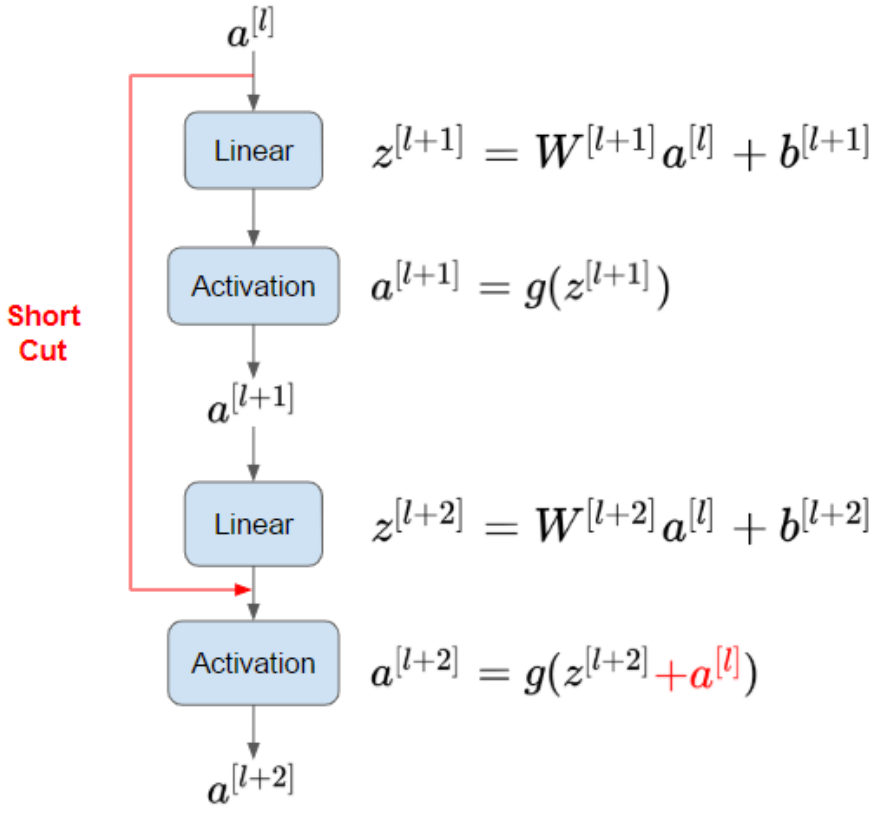
- 아주 깊은 신경망을 학습하지 못하는 이유
  - 경사가 소실되거나 폭발적으로 증가하기 때문

→ ResNet 에서는 스킵 연결로 이 문제를 해결

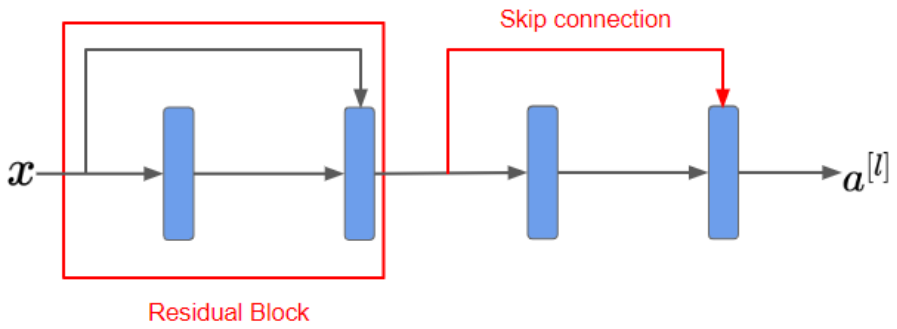
- 모든 층을 지나는 연산 과정: “main path”
  - 즉, a^{[l]}*a[l]* 의 정보가 a^{[l+2]}*a[l+2]* 로 흐르기 위해서는 모든 과정을 거쳐야 함



- main path
  - ResNet 에서는 z^{[l+2]} 에 비선형성을 적용해주기 전에 a^{[l]} 을 더하고 이것을 다시 비선형성을 적용



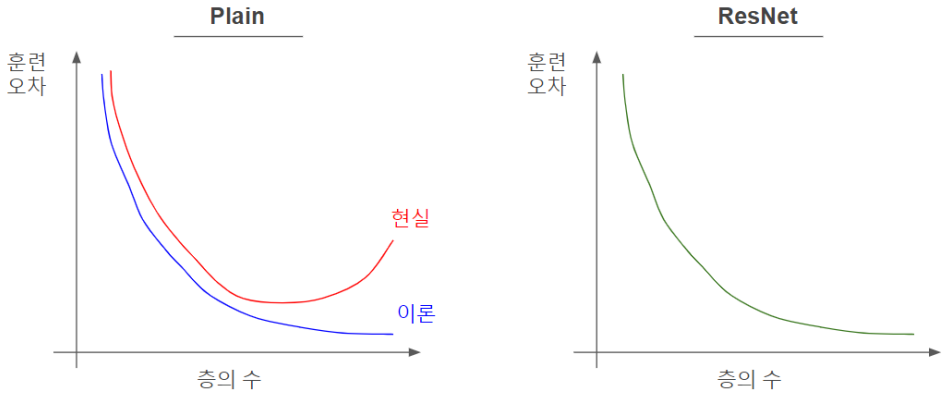
- **skip connection / short cut**
  - **잔여 블록**: a^{[l]}*a[l]* 를 더해서 다시 활성화 함수에 넣는 부분까지
  - Short cut / Skip connection 은 a^{[l]}*a[l]* 의 정보를 더 깊은 층으로 전달하기 위해 일부 층을 뛰어 넘는 역할
  - ResNet 은 여러개의 잔여 블록으로 구성
    - ResNet 을 만드려면 평형망에 스킵 연결.



- **residual block & skip connection**

- 경험적으로 층의 개수를 늘릴 수록 훈련 오류는 감소하다가 다시 증가
- 이론 상으로는 신경망이 깊어질 수록 훈련 세트에서 오류는 계속 낮아져야함.

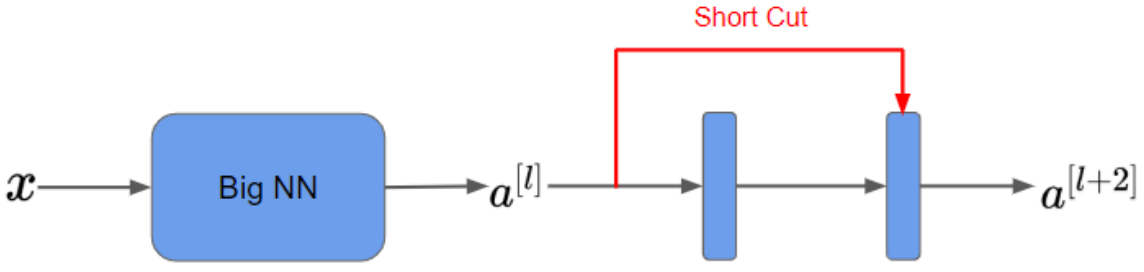
→ 하지만 ResNet 에서는 훈련오류가 계속 감소하는 성능



4. 왜 Resnets이 잘 작동?

- 경망의 깊이가 깊어 질 수록 훈련세트를 다루는데 지장

→ Resnets은 이 문제를 해결



$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$
$$= g(W^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

- **스킵 연결의 효용**
  - 큰 신경망에서 두개의 층을 더 추가하고 지름길을 연결.
    - 활성화함수: ReLU.
  - 스킵 연결을 더해준 출력값 a^{[l+2]}*a[l+2]* 은 g(z^{[l+2]} + a^{[l]}) = g(W^{[l+2]}a^{[l+1]} + b^{[l+2]}+

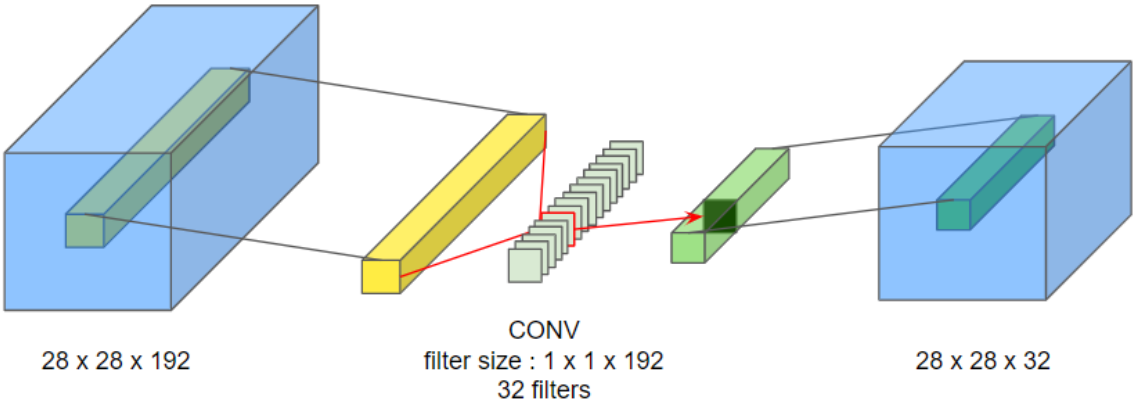
a^{[l]})$g(z[l+2]+a[l])=g(W[l+2]a[l+1]+b[l+2]+a[l])$ 로 쓸수 있다.

- W^{[l+2]}$W[l+2]$ 와 b^{[l+2]}$b[l+2]$ 의 값이 00 이 된다면, 위의 식은 a^{[l+2]} = g(a^{[l]}) = a^{[l]}$a[l+2]=g(a[l])=a[l]$ 으로 항등식이 됨.
- 항등식의 의미: 신경망으로 하여금 스킵 연결을 통해 두 층이 없는 더 간단한 항등식을 학습하여, 두 층 없이도 더 좋은 성능을 낼 수 있게 만든다는 것.
- z^{[l+2]}$z[l+2]$ 와 a^{[l]}$a[l]$ 이 같은 차원을 가져야 함.

→ 보통 동일합성곱 연산(출력 크기가 입력크게와 같기하는 합성곱연산)을 하거나 차원을 같게 만들어주는 행렬 W_sWs 를 잔여블록 앞에 곱해줘서 같게 만든다.

5. Network 속의 Network

- 합성곱 신경망을 구축할 때 1 x 1 합성곱은 매우 유용



- 195 개의 입력숫자가 32개의 1 x 1 필터와 합성곱을 하여 32 개의 출력 숫자가 됨.

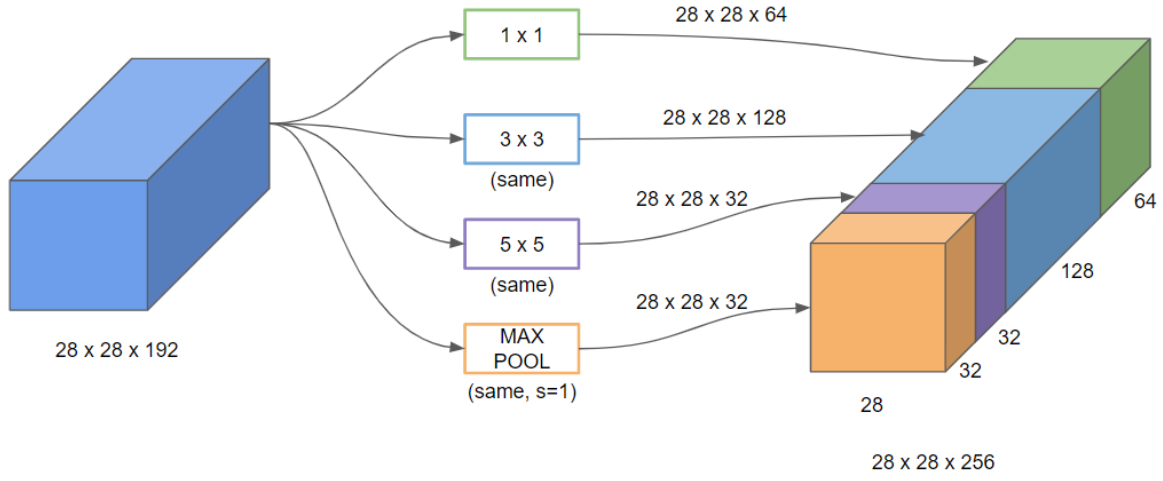
→ 입력 채널의 수만큼 유닛을 입력으로 받아서, 이들을 하나로 묶는 연산과정 통해, **출력채널의 수만큼 출력을 하는 작은 신경망 네트워크**로 간주 가능.

→ 네트워크 안의 네트워크

- 1x1 합성곱 연산을 통해 비선형성을 하나 더 추가해 복잡한 함수를 학습 시키기 가능
- 채널수를 조절 가능.

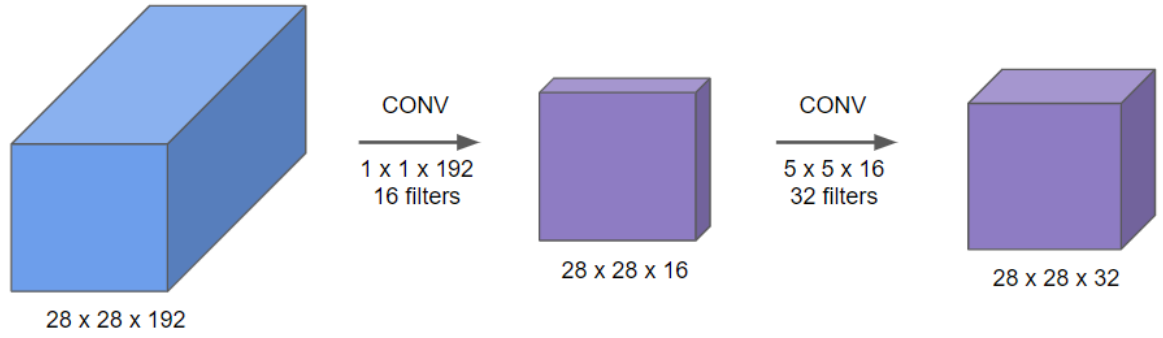
6. Inception 네트워크의 아이디어

- 인셉션 네트워크: 필터의 크기나 풀링을 결정하는 대신 전부 다 적용해서 **출력들을 합친 후, 네트워크 스스로 변수나 필터 크기의 조합을 학습하게 만드는 것**



- **인셉션 네트워크의 아이디어**
  - 문제: 계산 비용
  - 5 x 5 필터만 봐도 필요한 곱셈: 28 x 28 x 32 x 5 x 5 x 192 = 약 1억 2000 만개

→ 1 x 1 합성 곱으로 해결 가능



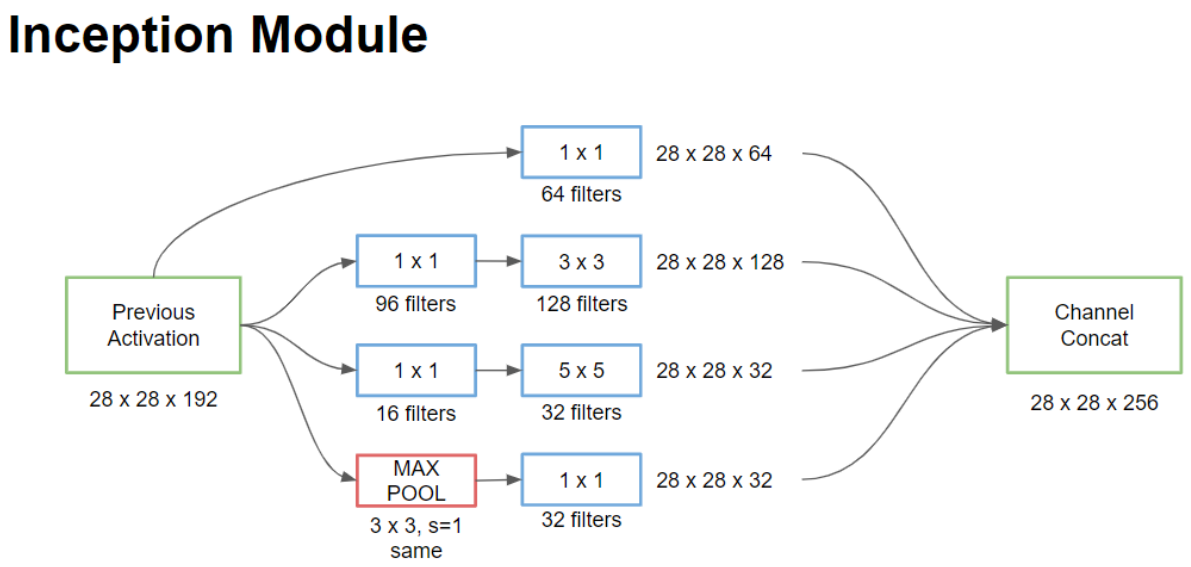
- 5 x 5 합성곱을 사용하기 전에 1 x 1 의 합성곱 연산을 통해 입력 이미지의 볼륨을 줄이는 작업
- 그 후에 다시 5 x 5 합성곱 연산
  - 계산 비용은 약 1240 만개
    - 1 x 1 합성곱: 28 x 28 x 16 x 1 x 1 x 192 = 약 240 만개
    - 5 x 5 합성곱: 28 x 28 x 32 x 5 x 5 x 16 = 약 1000 만개

→ 학습에 필요한 계산 비용이 1/10 수준으로 줄어든다.

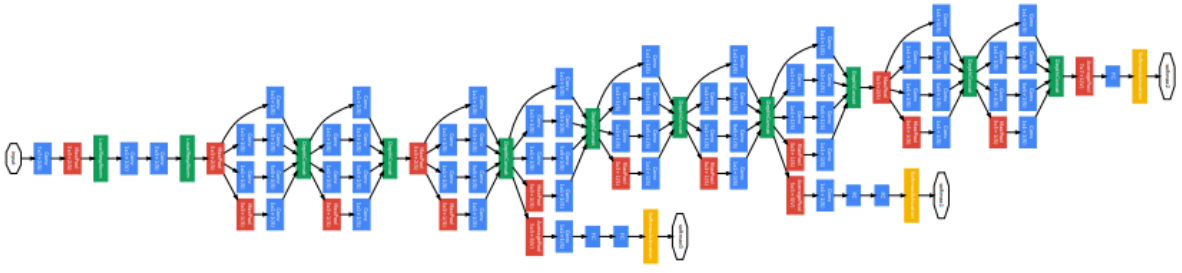
- 여기서 사용된 1 x 1 합성곱 층: **“병목 층”**

- 적절하게 구현시, 표현의 크기를 줄임과 동시에 성능에 큰 지장 없이 많은 수의 계산을 줄이기 가능

7. Inception 네트워크



- 인셉션 네트워크: 여러개의 인셉션 모듈로 구성



- 중간 중간에 차원을 바꾸기 위한 최대 풀링층을 포함해서 여러개의 인셉션 블록이 계속 반복됨

- 인셉션 네트워크는 구글넷