AI 프로그래밍

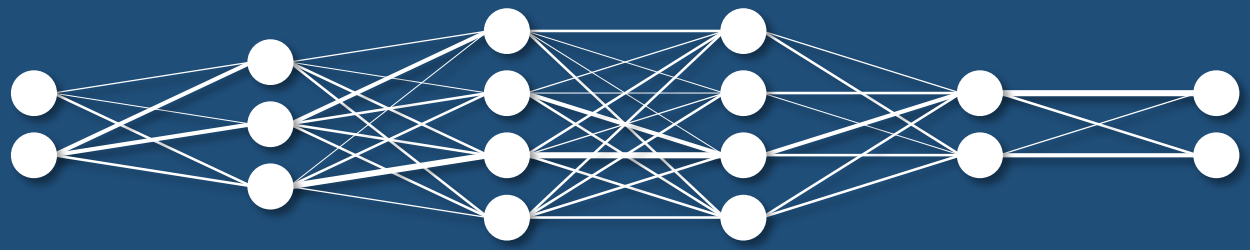
다양한 ANN Architecture

01

초창기 ANN의 Architecture와 한계

초창기 ANN의 Architecture와 한계

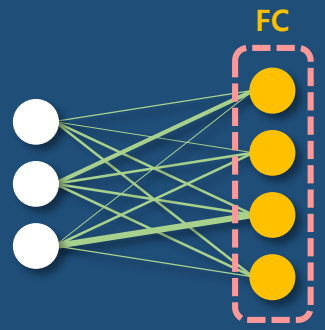
- 초창기 ANN의 architecture



  - ANN은 다수의 layer를 쌓아 구성
  - 각 layer의 node는 앞 layer와 뒤 layer의 모든 node와 edge로 연결
    - 그래프에서 모든 정점이 연결된 상태를 fully connected라고 함
    - ANN에서 인근 layer의 모든 node들과 edge로 연결된 layer를 fully connected layer라고 함

초창기 ANN의 Architecture와 한계

- Fully connected layer (FC)



  - 인근 layer의 모든 node와 edge로 연결된 layer
    - 이전 layer의 node가 m개, 현재 layer의 node가 n개 일 때, m × n 개의 edge가 있음
  - Node 개수를 임의로 정할 수 있는 장점이 있음
  - Node가 많아지면 edge의 개수가 급격히 많아지는 단점이 있음
    - ANN의 메모리 소모가 지나치게 많아질 수 있음

초창기 ANN의 Architecture와 한계

- Fully connected layer (FC)

256 × 256 × 3 = 196,608 nodes × 1,000 nodes

196,608,000 edges
196,608,000 × 4 bytes = 750 GB

- 일반적으로 ANN의 개별 parameter를 저장하기 위해 32bit (4 bytes) float를 사용
- 256 × 256 사이즈의 작은 RGB 이미지를 처리하는데, 한 layer에서만 750 GB가 필요함

02

이미지 처리와 Convolutional Layer

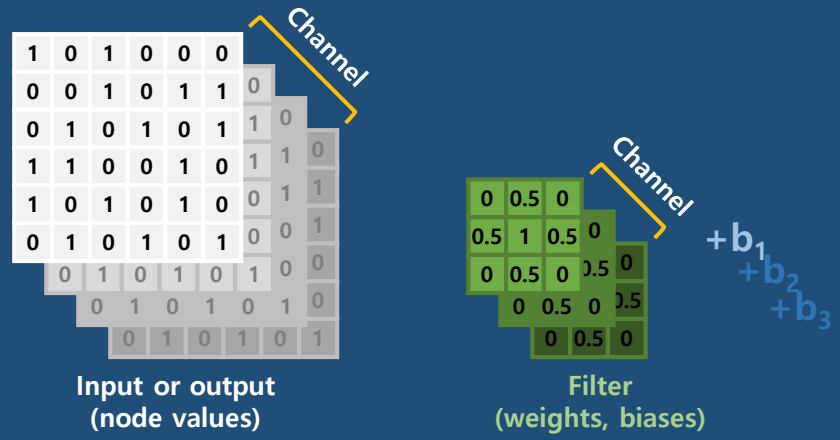
이미지 처리와 Convolutional Layer

- Convolutional layer (Conv)

- 픽셀 수가 많은 이미지를 처리하기 위해 개발된 layer
- 정해진 사이즈의 filter가 sliding window가 되어, 이미지를 훑어보면서 output 생성
  - Weight가 filter에만 존재
- Input 이미지 사이즈가 아무리 커도, weight의 수는 변하지 않음
- Convolutional layer로 이루어진 ANN을 convolutional neural network (CNN)이라 함

이미지 처리와 Convolutional Layer

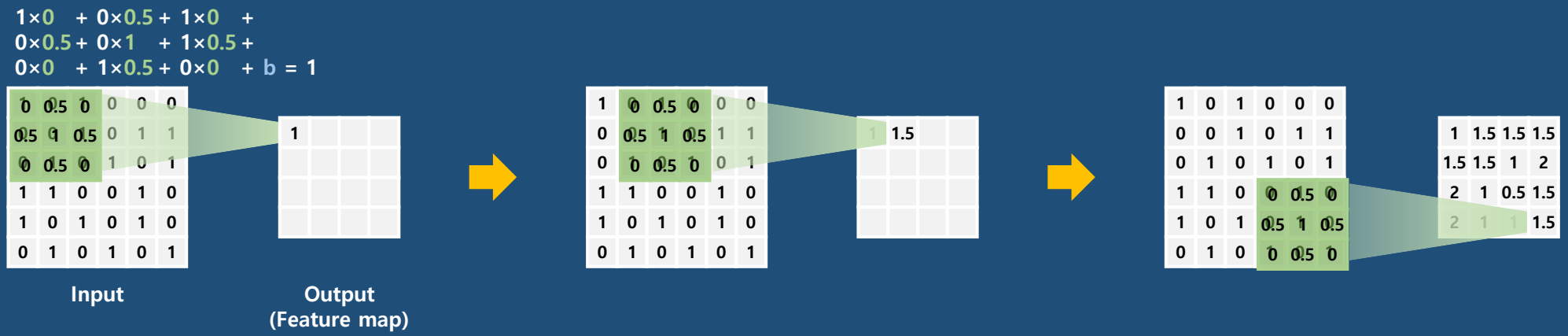
- Convolutional layer (Conv)
  - Convolutional filter



- Convolutional layer의 input, output, 그리고 filter는 width, height, channel로 구성
- Filter의 각 칸에는 하나의 weight가, 각 channel에는 하나 bias가 존재함
- Filter는 일반적으로 홀수 × 홀수 사이즈의 정사각형

이미지 처리와 Convolutional Layer

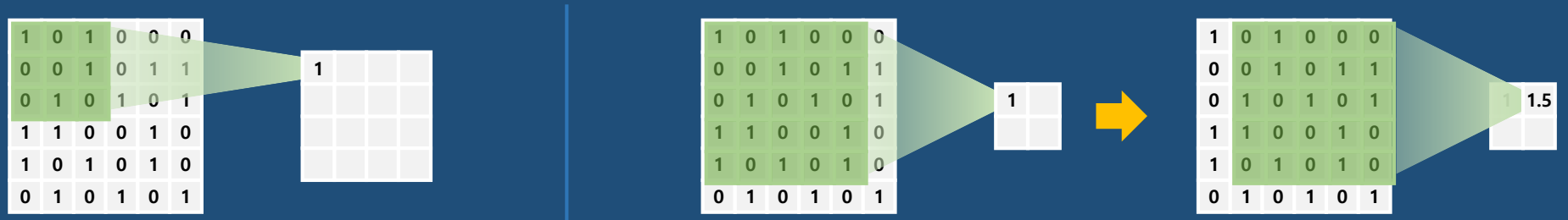
- Convolutional layer (Conv)
  - Convolutional filter의 작동 방식



- 실제로는 차례대로 훑어보지 않고, feature map의 모든 node를 동시에 계산함
- 단 weight는 동일하게 적용되므로, weight를 저장할 메모리 공간이 크게 절약됨
  - 공통의 weight를 사용하는 것을 weight sharing이라 함

이미지 처리와 Convolutional Layer

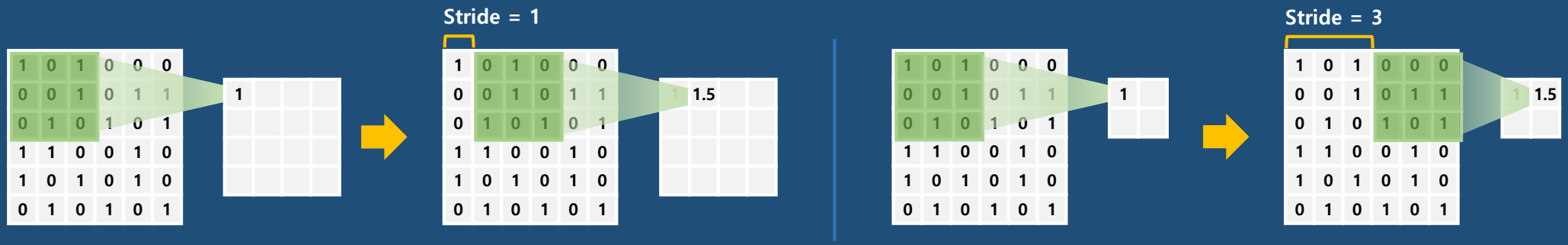
- Convolutional layer (Conv)
  - Convolutional filter의 작동 방식



- Filter의 width와 height가 클 수록 feature map의 width와 height가 작아짐
  - Filter는 input 이미지 밖으로 나갈 수 없음

이미지 처리와 Convolutional Layer

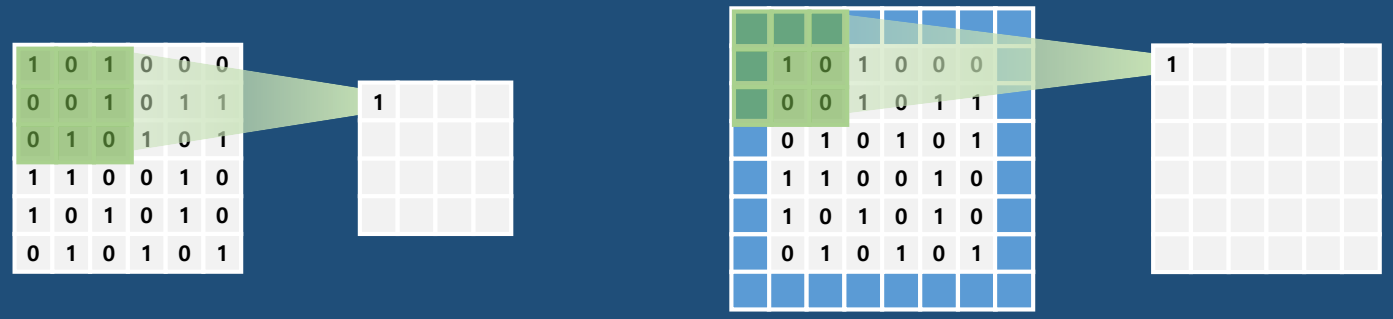
- Convolutional layer (Conv)
  - Stride : Filter를 배치하는 간격



  - Stride가 클 수록 feature map의 width와 height가 작아짐

이미지 처리와 Convolutional Layer

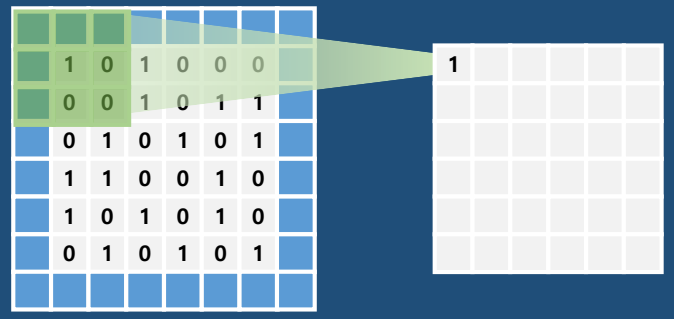
- Convolutional layer (Conv)
  - Padding : Input 주변에 추가 픽셀을 배치



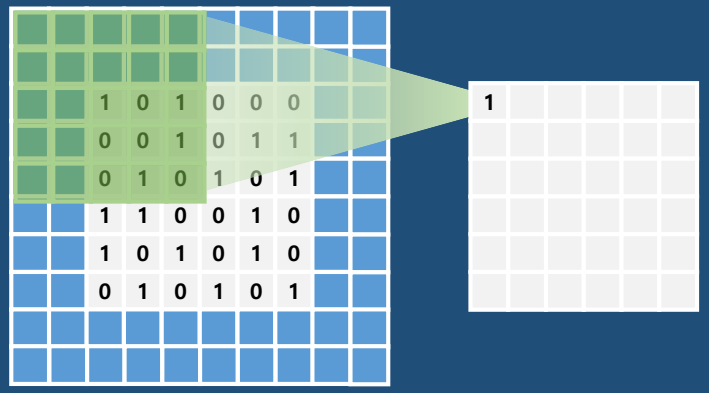
  - Feature map의 width와 height가 줄어드는 것을 방지하고 싶을 때 사용

이미지 처리와 Convolutional Layer

- Convolutional layer (Conv)
  - Padding : Input 주변에 추가 픽셀을 배치



Padding size = 1

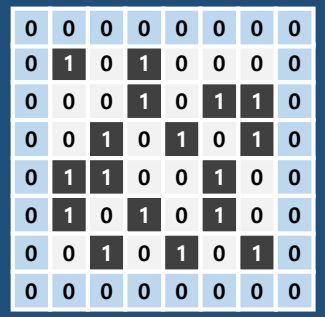


Padding size = 2

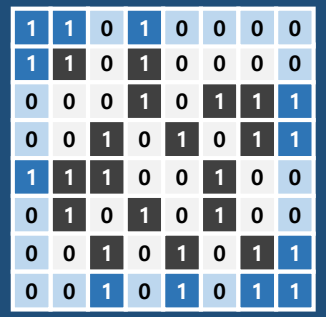
  - Filter의 width, height가 큰 만큼, padding size도 늘어나야 feature map의 width 및 height를 유지할 수 있음

이미지 처리와 Convolutional Layer

- **Convolutional layer (Conv)**
  - **Padding : Input 주변에 추가 픽셀을 배치**



Zero padding
(padding size = 1)

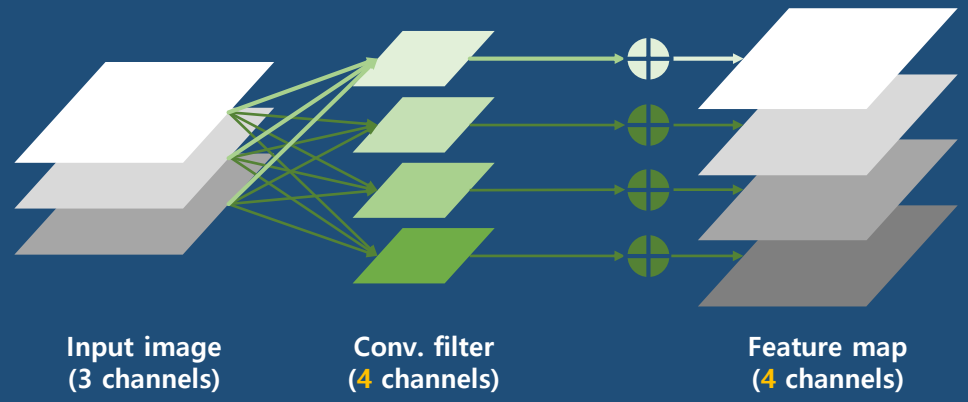


Same padding
(padding size = 1)

- **Zero padding** : 추가 픽셀에 0을 채움
- **Same padding** : 추가 픽셀에 테두리 값을 복사하여 채움

이미지 처리와 Convolutional Layer

- Convolutional layer (Conv)
  - Convolutional layer의 channel
    - Filter의 한 channel이 input 이미지의 모든 channel에 적용된 후, pixel-wise로 더해져서 feature map의 한 channel이 됨
    - Feature map은 filter와 같은 channel 수를 가짐



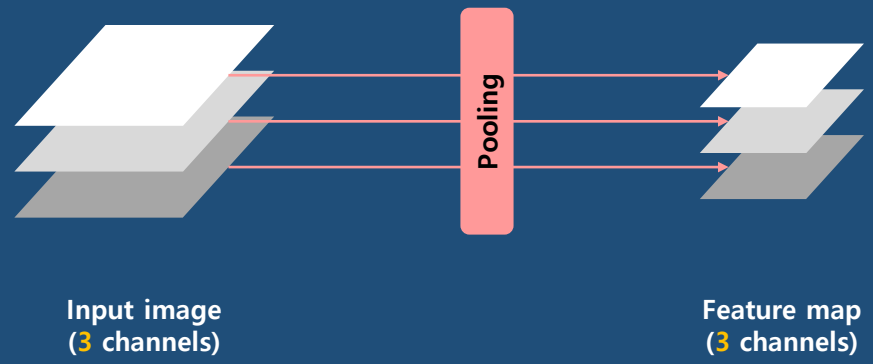
이미지 처리와 Convolutional Layer

- Pooling layer
  - Pooling : Weight 없이, input의 사이즈를 크게 줄여 압축할 때 활용

- 압축 방법에 따라 max pooling, mean pooling이 있음
- Pooling filter는 일반적으로 짝수 × 짝수의 정사각형으로 구성
- Stride를 조정하여 output의 차원을 조정할 수 있음

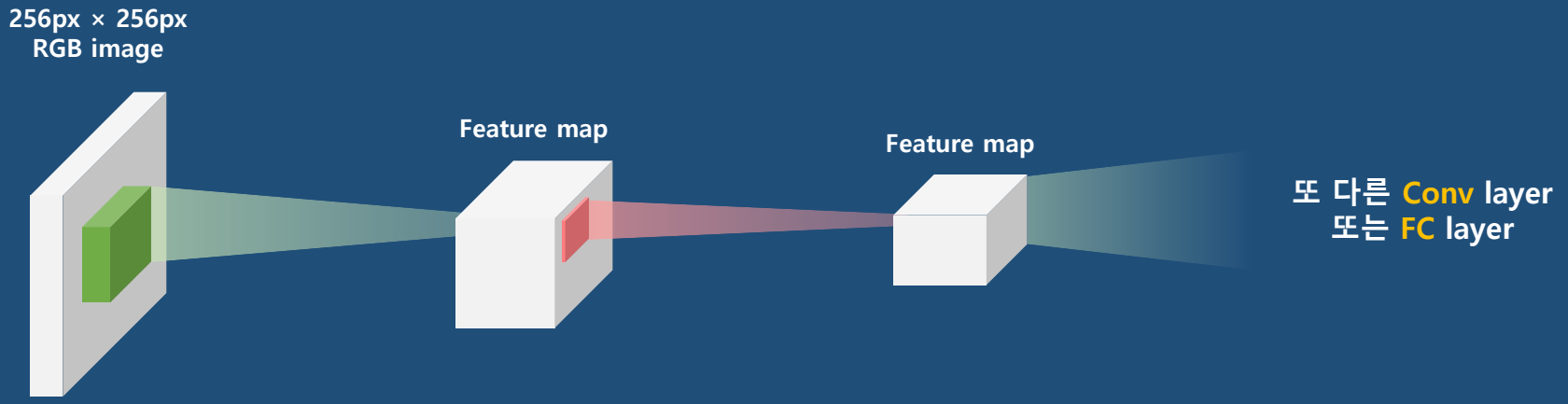
이미지 처리와 Convolutional Layer

- Pooling layer
  - Pooling layer의 channel
    - Input의 각 channel에 개별적으로 수행됨
    - Input의 channel이 유지됨



이미지 처리와 Convolutional Layer

- Convolutional neural network (CNN)
  - 일반적인 CNN의 구성



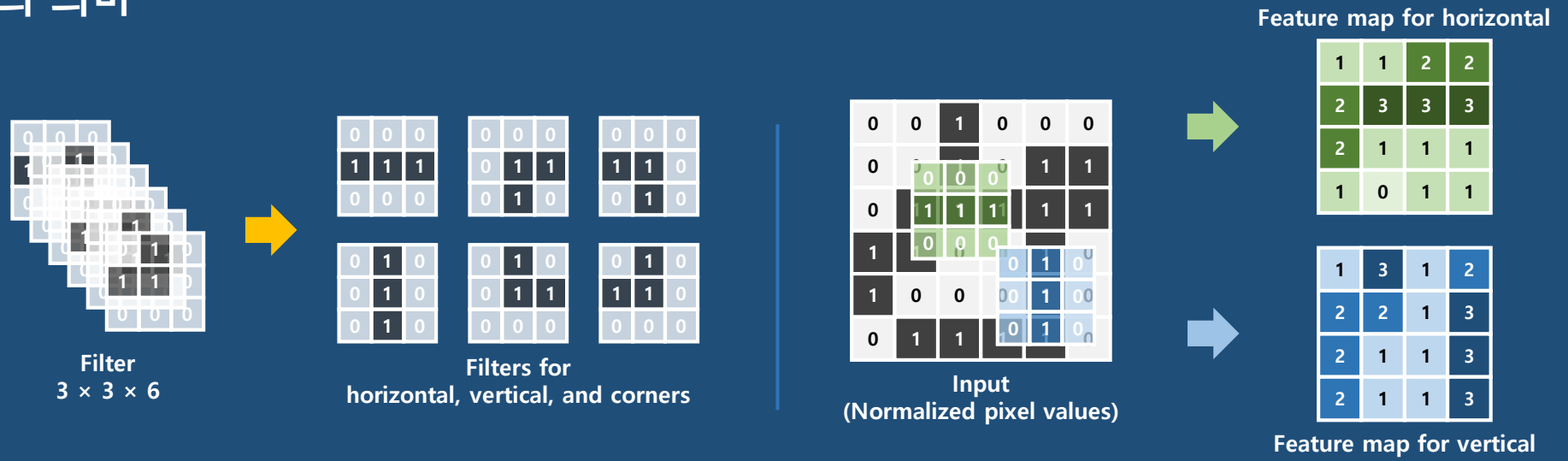
- Input : 256 × 256 × 3
- Filter : 5 × 5 × 6
- Stride = 2
- padding size = 2

- Output : 128 × 128 × 6
- Pooling : 2 × 2
- Stride = 2

- Output : 64 × 64 × 6

이미지 처리와 Convolutional Layer

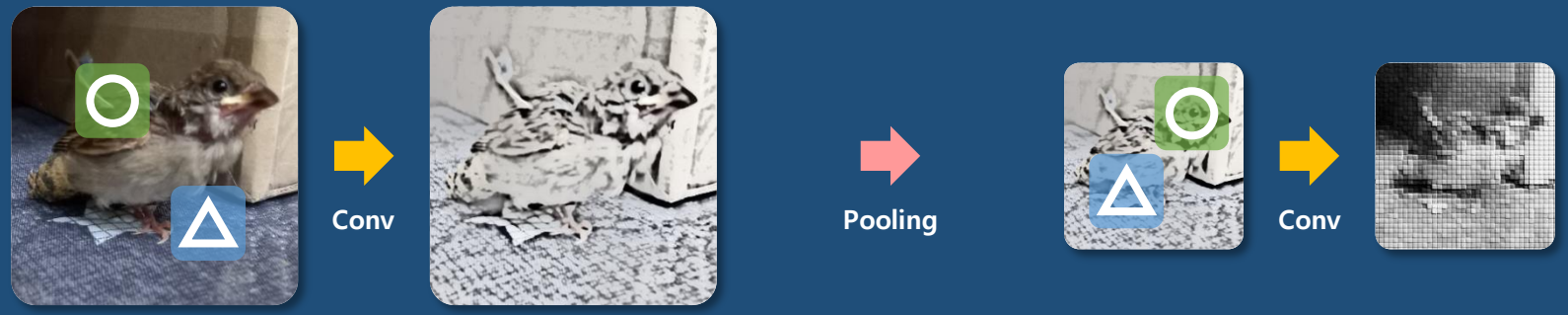
- Convolutional neural network (CNN)
  - Filter의 의미



- Filter가 이동하며 input 이미지를 보기 때문에, 이미지 곳곳에서 visual feature를 감지함
- Input의 어느 위치에서도 특정 feature를 찾아 내는 것을 translation invariant라 함

이미지 처리와 Convolutional Layer

- Convolutional neural network (CNN)
  - Pooling의 의미



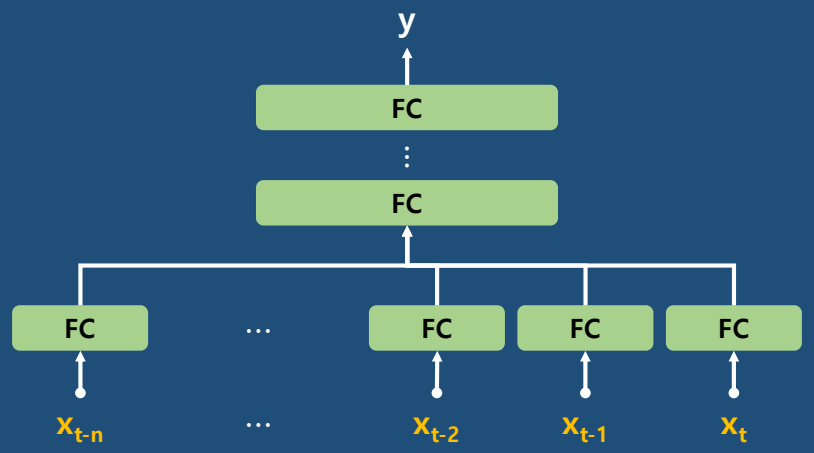
- Input를 작게 만들면, filter 사이즈는 유지하면서 더 넓게 보는 효과를 가져옴
  - Filter 사이즈를 늘리지 않으면 weight 수가 늘어나지 않게 할 수 있음
- Input의 사이즈(scale)가 다양할 때도 특정 feature를 찾아 내는 것을 scale invariant라 함

03

Sequential 데이터와 LSTM

Sequential 데이터와 LSTM

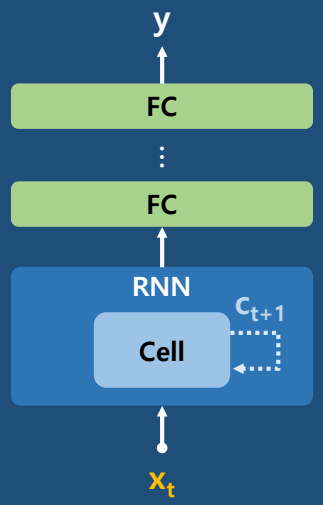
- 초창기 ANN의 sequential data 처리 방법



- 기존의 ANN은 뉴런 값을 계속해서 기억할 수 없음
- 따라서 연속적인 데이터가 필요한 경우 이를 모두 동시에 input으로 주어야 함
  - Sequence가 길어질 수록 weight가 많아짐

Sequential 데이터와 LSTM

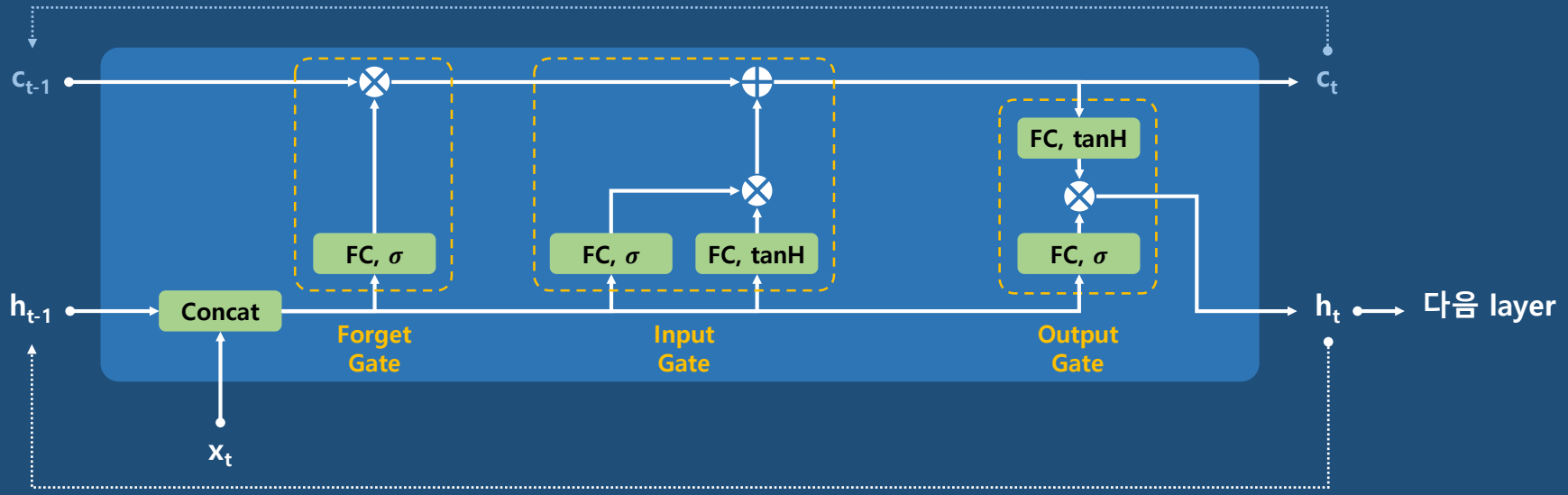
- Recurrent neural network (RNN)



- 이를 해결하기 위해 뉴런 값을 기억할 cell이란 개념을 추가
- 이러한 형태의 ANN을 recurrent neural network (RNN)이라 함

Sequential 데이터와 LSTM

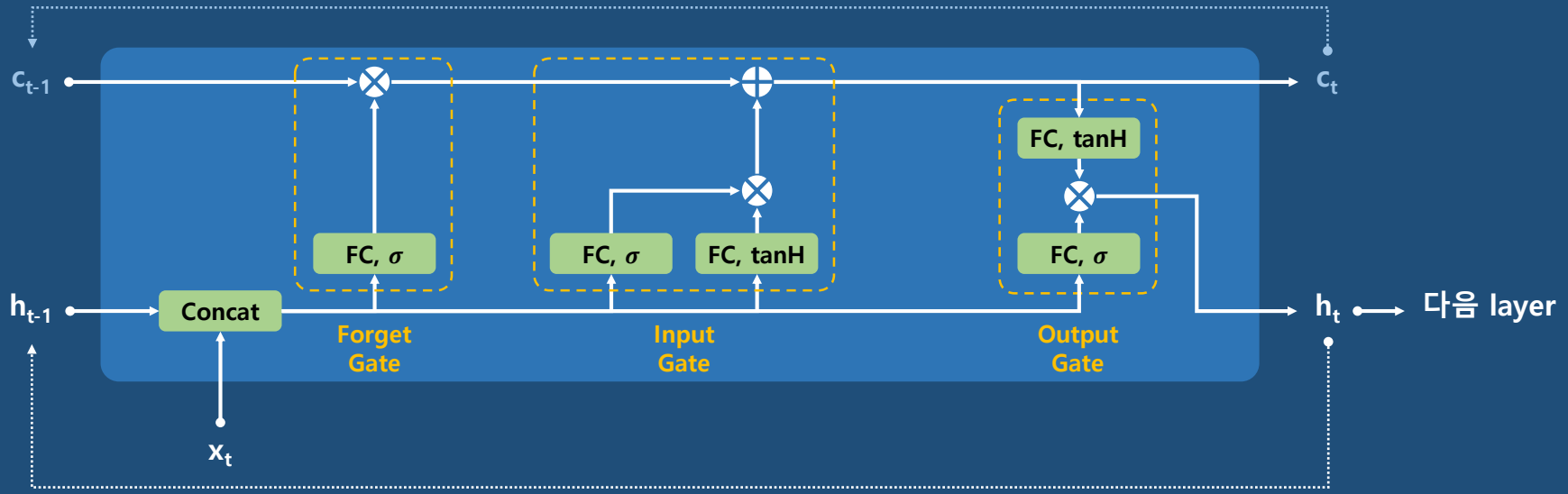
- Long short-term memory (LSTM)



- Recurrent neural network (RNN)의 일종으로 내부적으로 기억(cell, c_t)을 가지는 layer
- 내부적으로 다양한 게이트로 구성됨

Sequential 데이터와 LSTM

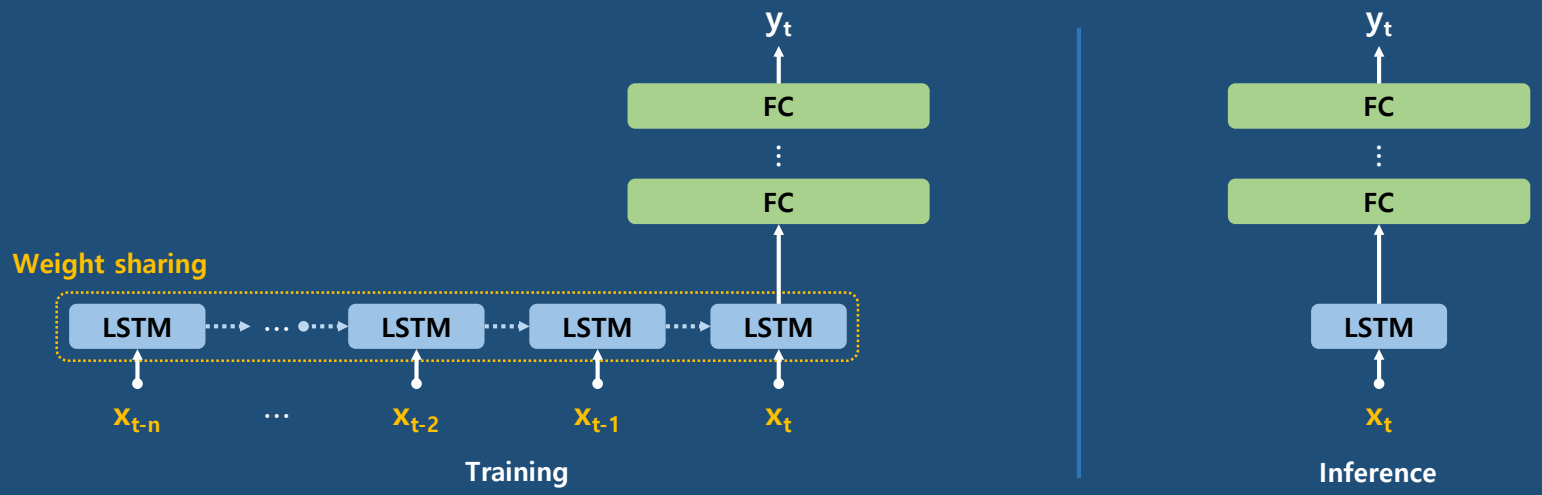
- Long short-term memory (LSTM)



- Forget gate : 이전 기억 c_{t-1}에 sigmoid output을 곱하여 보존 정도를 결정
- Input gate : y_{t-1}, x_t를 종합하여 현재 기억 c_t를 생성
- Output gate : c_{t-1}, y_{t-1}, x_t를 종합하여 최종 output y_t를 생성, 다음 layer로 전달

Sequential 데이터와 LSTM

- LSTM의 training과 inference



- RNN은 training과 inference의 ANN architecture가 다름
- Training시에는 cell을 포함한 layer를 sequence 길이만큼 전개함
  - 전개한 layer는 모두 같은 weight를 공유함

04

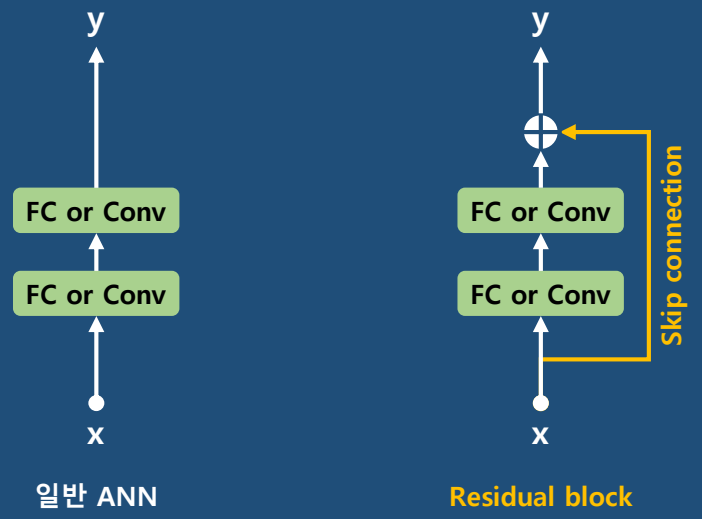
딥러닝을 위한 ANN

딥러닝을 위한 ANN

- Gradient Vanishing
  - 복잡한 문제를 해결하기 위해선 많은 hidden layer를 쌓아야 함
    - Hidden layer가 많을 수록 ANN이 deep하다고 표현함
  - Hidden layer가 너무 많은 경우 input layer에 가까워 질 수록 gradient가 0에 수렴할 수 있음
    - 따라서, input layer에 가까운 layer들은 training이 일어나지 않음
    - 복잡한 문제 해결을 위해 hidden layer가 많이 쌓아도 소용이 없게 됨
  - 이러한 문제를 해결하기 위해 ResNet과 DenseNet 개발 됨

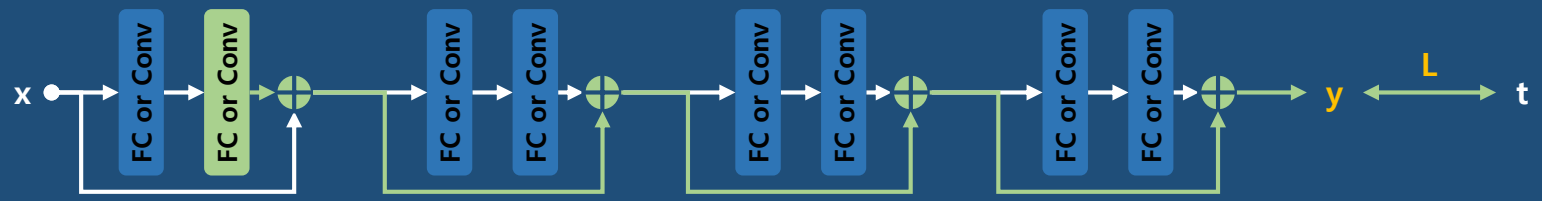
딥러닝을 위한 ANN

- ResNet (Residual neural network)
  - Residual block을 통해 인접하지 않은 layer들을 직접 연결(skip connection)



딥러닝을 위한 ANN

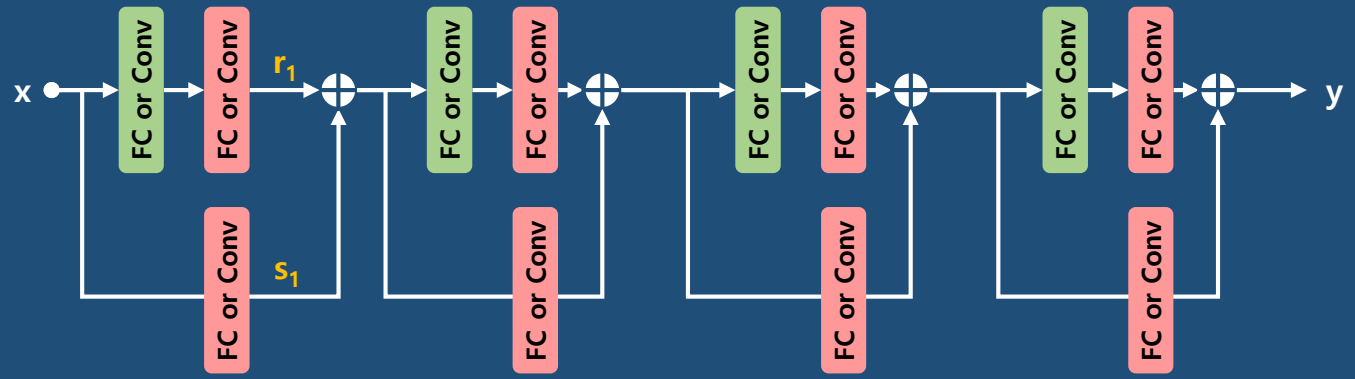
- ResNet (Residual neural network)
  - + 연산을 통해 이전 layer의 값이 더해짐
    - Gradient를 구하기 위해 loss 함수 L을 미분하면
      멀리 떨어진 layer의 weight의 영향력이 그대로 유지됨



  - Input layer에 가까워지더라도 Gradient가 0으로 수렴되는 것을 방지함

딥러닝을 위한 ANN

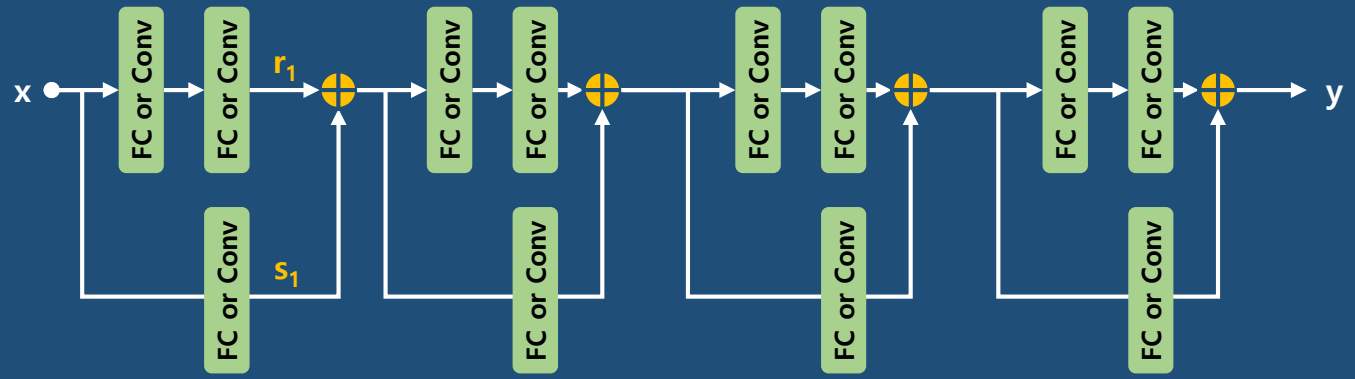
- ResNet (Residual neural network)
  - + 연산을 하려면 차원이 같아야 하므로, 다음 residual block으로 넘어갈 때 유의해야 함
    - 아래 그림에서 r_1과 s_1의 차원이 같아야 더할 수 있음



  - Residual block이 넘어 갈 때 차원을 줄여야 한다면
    skip connection에서도 residual block의 output과 같은 차원을 줄여야 함

딥러닝을 위한 ANN

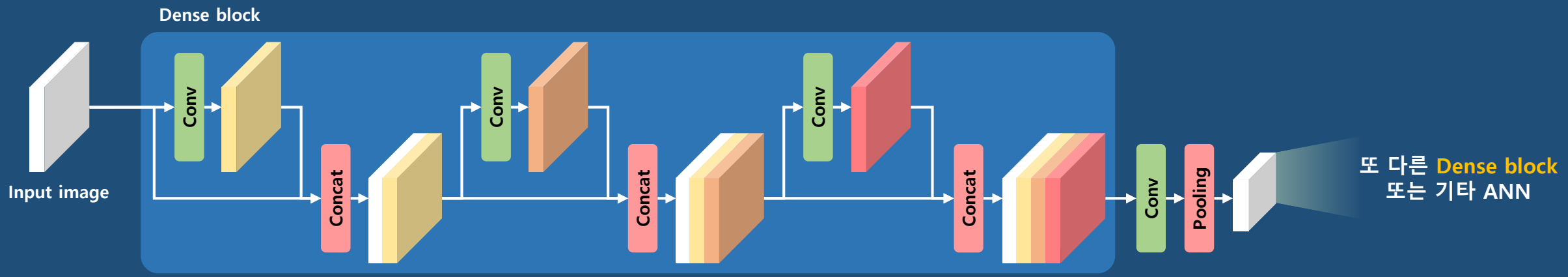
- ResNet (Residual neural network)
  - 다음 Residual block의 input이, + 연산을 통해 데이터가 뭉개짐
    - 아래 그림에서 r_1과 s_1가 더해지면서 그 원래 값을 잃어버림



  - 이를 해결하기 위해 DenseNet이 개발됨

딥러닝을 위한 ANN

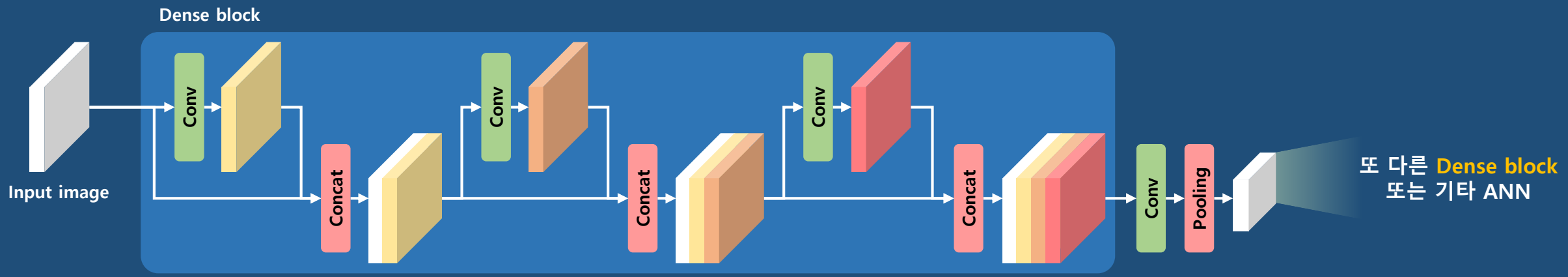
- DenseNet
  - 이전 input을 더하는 것이 아니라, concatenate하여 유지시키는 CNN



  - Convolutional layer만 사용 가능
  - Concatenate로 인해 feature map의 차원이 커지더라도, filter는 정해져 있으므로 weight 수는 일정함

딥러닝을 위한 ANN

- DenseNet
  - Concatenate를 위해서 dense block의 모든 feature map들은 input과 width 및 height가 같아야 함



  - 연산 량이 너무 커지는 것을 막기 위해 각 feature map의 channel 수는 적어야 함

딥러닝을 위한 ANN

- ResNet과 DenseNet의 비교
  - ResNet
    - 장점
      - + 연산을 사용하므로, residual block output의 dimension (node 개수)이 늘어나지 않음
    - 단점
      - + 연산으로 인해 residual block output이 훼손 됨
  - DenseNet
    - 장점
      - Concatenate를 사용하므로 input 및 이전 feature map이 보존됨
    - 단점
      - Concatenate로 인해 dense block 안에서 feature map의 channel이 너무 커질 수 있음
      - Fully connected layer를 사용할 수 없음

감사합니다

Fin